Arquitetura de Computadores: educação, ensino e aprendizado

Editora
Sociedade Brasileira de Computação - SBC

Organizadores
Carlos Augusto Paiva da Silva Martins
Philippe Olivier Alexandre Navaux
Rodolfo Jardim de Azevedo
Sérgio Takeo Kofuji

Arquitetura de Computadores: educação, ensino e aprendizado

Organizadores
Carlos Augusto Paiva da Silva Martins (PUC Minas)
Philippe Olivier Alexandre Navaux (UFRGS)
Rodolfo Jardim de Azevedo (UNICAMP)
Sérgio Takeo Kofuji (USP)

ISBN: 978-85-7669-263-8
Ano de publicação - 2012

Editora
Sociedade Brasileira de Computação - SBC

Sumário

Lista de autores dos capítulos

Alberto Hideki Ueda (capítulo 4)
Alfredo Goldman (capítulo 4)
André Luis Alice Raabe (capítulo 6)
Anfranserai M. Dias (capítulo 2)
Angelo A.Duarte (capítulo 2)
Antonio L. Apolinário Jr. (capítulo 2)
Camila Mari Matsubara (capítulo 4)
Carlos Augusto Paiva da Silva Martins (capítulo 7)
Cesar Albenes Zeferino (capítulo 6)
Durgam Vahia (capítulo 5)
Edson Horta, Eduardo Lima (capítulo 5)
Edward David Moreno (capítulo 5)
Filippo Valiante Filho (capítulo 5)
Henrique Cota de Freitas (capítulo 3)
João Paulo dos Santos Mota (capítulo 4)
Jussara M. Kofuji (capítulo 5)
Leonardo A. G. Garcia (capítulo 5)
Maicon Carlos Pereira (capítulo 6)
Márcio Lobo Netto (capítulo 5)
Marco Antonio Zanata Alves (capítulo 3)
Paulo Viniccius Vieira (capítulo 6)
Pedro Henrique Borges de Almeida (capítulo 1)
Philippe Olivier Alexandre Navaux (capítulo 3)
Rodolfo Jardim de Azevedo (capítulo 1)
Sergio T. Kofuji (capítulo 5)
Tiago de Oliveira (capítulo 2)
Wagner L. A. de Oliveira (capítulo 2)
Wang J Chau (capítulo 5)

Capítulo 1

Aplicação de Tablet PCs no Ensino de Arquitetura de Computadores

Pedro Henrique Borges de Almeida (pedrohbalmeida@gmail.com)

Rodolfo Jardim de Azevedo (**rodolfo@ic.unicamp.br**)

Instituto de Computação – UNICAMP - Av. Albert Einstein, 1251 - Sala 10 - Cidade Universitária - 13083-852 ou Caixa Postal 6176 - Cidade Universitária - 13083-970 Telefone: +55 (19) 3521-5857 - Fax: +55 (19) 3521-5847

Resumo

Não é novidade alguma a introdução de dispositivos computacionais no ambiente de sala aula. Sejam eles notebooks, PDAs, celulares ou qualquer outro tipo de dispositivo móvel, a eles é dado o crédito de permitir que o ambiente de ensino seja melhor aproveitado. Dentre esses dispositivos móveis destaca-se o Tablet PC, que possui grande mobilidade, poder computacional semelhante ao dos notebooks modernos, tela dotada de interface pen-based e rede wireless. Tem-se assim um hardware bastante promissor para o desenvolvimento de aplicativos e práticas que tirem proveito de diversos paradigmas de ensino, como o ensino ativo e colaborativo, cujos benefícios já foram demonstrados na literatura.

A UNICAMP possui uma sala de aula com 21 Tablet PCs e infra-estrutura wireless. Com esses equipamentos foi possível a montagem de um ambiente de ensino colaborativo e ativo, no qual o professor pode interagir com os estudantes através de exercícios e da troca de anotações digitais. Além disso, foi possível que o docente gravasse as aulas ministradas em vídeo, disponibilizando-as aos alunos após seu término.

Esse capítulo irá retratar as bases da metodologia utilizada e quais benefícios se espera obter destas práticas. O ambiente utilizado será descrito, incluindo modificações feitas tanto no curso quanto nas ferramentas utilizadas, enfatizando como a introdução do Tablet PC em sala de aula influenciou o comportamento dos alunos.

1. Introdução

A introdução de dispositivos computacionais em sala de aula, associada a uma metodologia adequada, pode resultar numa melhor qualidade de ensino? Se sim, em que áreas? O objetivo deste capítulo é responder tais questões, tratando da introdução de Tablet PCs num curso de Arquitetura de Computadores.

Dentre o grande número de equipamentos eletrônicos que estão sendo lançados recentemente, um deles em especial está chamando a atenção de educadores em todo o mundo: o Tablet PC. O grande diferencial desse equipamento é a associação entre poder computacional, semelhante ao de notebooks, mobilidade e interface *pen-based*. A possibilidade de escrita precisa, direto na tela, faz do Tablet PC o equipamento ideal em aulas que tenham uma carga densa de símbolos, diagramas e equações, dentre outros que são de difícil reprodução com o uso de mouse e teclado.

Com isso, temos uma interface que possibilita que os alunos exerçam toda a sua criatividade, ao mesmo tempo em que utilizam as facilidades providas por um computador. Com isso, abre-se campo para o desenvolvimento de uma série de novas metodologias de ensino que explorem correntes pedagógicas de eficácia comprovada, porém, de difícil implantação.

A metodologia aqui discutida refere-se às praticas de ensino ativo e colaborativo, onde os alunos são instigados a resolverem exercícios durante a aula, saindo assim do papel de meros espectadores e participando melhor da aula. Com o uso de Tablet PCs, pode-se aplicar essa metodologia de forma a atingir os seguintes objetivos:

- Melhorar a forma de apresentação do conteúdo;
- Aumentar a participação dos estudantes durante a aula, fazendo com que alunos tímidos ou com dificuldades no aprendizado sejam mais participativos;
- Fornecer uma forma de retorno rápido para o professor;
- Fornecer material didático extra aos alunos, em especial os *slides* do professor com anotações e vídeos das aulas.

Todas essas mudanças serão discutidas no decorrer deste capítulo em detalhes, mostrando as dificuldades envolvidas em sua aplicação, assim como o uso dessa metodologia afetou o curso de Arquitetura de Computadores ministrado na Unicamp.

2. Utilização do Tablet PC no Ensino de Arquitetura de Computadores

Quando se fala em ensino ativo e colaborativo, evoca-se uma idéia geral que tem como principal argumento a transferência da maioria da responsabilidade do aprendizado para o aluno, fazendo com que ele tenha um papel crucial no seu processo de aprendizagem. Para tal objetivo, o instrutor deverá modificar a estrutura tradicional das aulas magnas, de forma a incluir atividades práticas que façam com que os alunos exercitem suas habilidades comportamentais e cognitivas[3].

A implementação de tais ambientes pode se dar de diversas formas, as quais são motivo constante de pesquisa e controvérsia entre os educadores. Questões versando sobre quando e como aplicar os exercícios são constantes, e práticas similares normalmente não obtém o mesmo resultado, quando aplicadas a cursos diferentes. Outra questão pertinente é sobre os tipos de exercícios que devem ser aplicados, sendo os mais comuns na literatura as discussões entre os alunos, a resolução de problemas ou até mesmo o estudo de exemplos resolvidos.

Não cabe no escopo deste trabalho um levantamento mais detalhado dos resultados obtidos em diversas áreas - o leitor mais interessado poderá consultar a seção de referências para encontrar material específico. Serão discutidos, aqui, a técnica que foi utilizada no curso e os resultados encontrados em trabalhos correlatos, que motivaram e justificaram a aplicação desta no curso de Arquitetura de Computadores, no 2º semestre do ano de 2008.

Além do ambiente de ensino ativo e colaborativo, também colocou-se em prática a idéia de gravação de *screencasts*, que nada mais são do que um vídeo contendo a fala do docente durante a aula, sincronizadas com a apresentação de seus *slides*. É notório citar aqui que a intenção não é a geração de material que substitua a aula ou para aplicação em ensino à distância, mas sim a obtenção de um conteúdo de apoio, que permita aos alunos tirarem dúvidas pontuais surgidas durante a aula ou que alunos faltantes possam ter acesso ao conteúdo produzido em sala de aula. Tem-se, portanto, a geração de um material que atende a um nicho muito específico de alunos, sendo estes os matriculados na própria disciplina. Para que esta idéia se torne factível, é necessário que a gravação dos *screencasts* não acarrete em mudanças bruscas na metodologia de apresentação e ensino do docente, bem como não produza a necessidade de um esforço extraclasse muito grande, como a necessidade de manipulação e edição do conteúdo gerado.

Como essas idéias são de certa forma independentes umas das outras, isto é, pode-se montar um ambiente ativo e colaborativo sem gravar *screencasts* e vice-versa, alguns leitores podem achar interessantes ou passíveis de implantanção algumas das técnicas e ferramentas aqui apresentadas. Por isto, dividimos o conteúdo desse capítulo em quatro subseções:

- A primeira irá discutir o Tablet PC em mais detalhes, além de mostrar como esse equipamento pode ser utilizado para a criação de ambientes de ensino ativo e colaborativo, assim como utilizado para a gravação de *screencasts*;

- A segunda subseção aborda as principais mudanças realizadas no referido curso de Arquitetura de Computadores, de forma que esse utilizasse os Tablet PCs para as atividades colaborativas e a gravação de *screencasts*;
- A terceira descreve os resultados encontrados no curso em que a metodologia foi utilizada;
- A quarta apresenta as conclusões obtidas pelos pesquisadores, com base nos resultados apurados.

2.1. *Tablet PC*: o equipamento

Da vasta gama de dispositivos móveis que surgiram na última década, pode-se destacar o Tablet PC como um equipamento que reúne um conjunto de funcionalidades considerada bem próxima do ideal, para aplicação em ambientes educacionais. Grosso modo, ele pode ser definido como um notebook com alta mobilidade, porém, dotado de interface *pen-based* - seu grande diferencial. Além disso, fatores como baixo peso, alta mobilidade, compatibilidade com softwares de PC e longa duração da bateria também são diferenciais desse dispositivo.

A interface *pen-based* do Tablet PC permite escrita diretamente na tela, como alguns modelos de *smart phones* ou *PDAs*, porém os modelos que realmente se destacam são aqueles que contam com um sensor magnético, chamado de *digitizer*, para detecção da escrita, ao invés de sensores de pressão resistivos ou capacitivos, diretamente embutidos na tela. A tecnologia do *digitizer* utilizado é semelhante à tecnologia empregada em Tablets de mesa, sendo que a tela é dotada de um receptor de RF, capaz de captar a posição da caneta numa taxa de 100hz com uma resolução de 600 pontos por polegada [7]. Em geral, há uma leve perda de precisão nos cantos da tela, a qual é imperceptível na maioria dos modelos. A tela dos Tablet PCs mais modernos pode ser girada em 180º e dobrada por cima de seu teclado, sendo essa forma chamada de *slate*. A figura 1 mostra um Tablet PC nas suas duas formas, e a figura 2 mostra um esquemático do hardware da interface *pen-based*.

Figura 1 – Tablet PC na forma normal e na forma *slate*

Além da precisão inerente ao hardware, há um tratamento especial realizado pelo sistema operacional *Windows* [5]. Os pontos da localização da caneta não são simplementes amostrados, como é comum na implementação de outros dispositivos. Há diversos algoritmos de tratamento na escrita, transformando o conjunto desses pontos em diversas curvas, fazendo com que a qualidade de escrita fique ainda

melhor, o que possibilita o desenho de símbolos complicados de forma extremamente precisa. O reconhecimento de escrita também é bem preciso, porém não foi utilizado durante a execução deste trabalho.

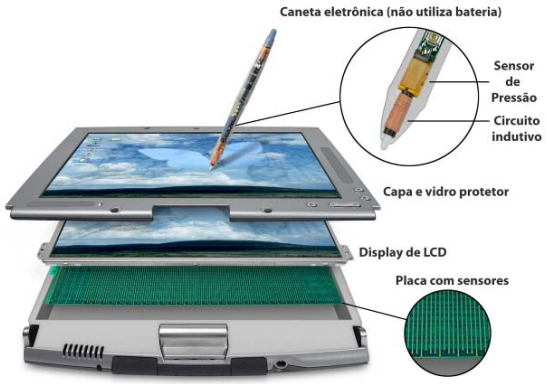


Figura 2 – Esquemático do hardware de escrita

Grande parte do esforço de projeto do hardware da interface *pen-based* dos Tablet PC*s* foi voltada para fazer com que a sensação de escrita seja a mais parecida possível com a escrita numa folha de papel convencional. As interfaces *pen-based* tradicionais existentes nos *PDAS* e *smart phones* não abordam esse problema com o mesmo cuidado. Isso faz com que, ao contrário desses dispositivos onde o foco principal fica no custo da tela e no consumo de energia, a escrita em Tablet PC*s* equipados com os *digitizers* seja uma experiência radicalmente diferente da escrita nos dispositivos previamente citados, sendo que são tomados cuidados com o material de revestimento da tela, para aproximar a sensação de rugosidade encontrada durante a escrita em papel convencional e com o sensor de pressão da caneta, que funciona de forma muito precisa. Além disso, o tamanho de tela dos Tablet PC*s* é semelhante ao de uma folha de tamanho A4, eliminando assim restrições quanto à área de escrita diminuta encontrada nos dispositivos menores.

O poder de processamento dos Tablet PCs atuais é semelhante ao de notebooks convencionais, sendo a única diferença o fato de ser comum a ausência de drives de CD/DVD nos Tablet PCs, a fim de se obter maior mobilidade e economia de energia. A tabela 1 ilustra as principais características da configuração de um modelo de Tablet PC da HP (tx1040br), vendido por cerca de R$ 3.500,00 no mercado nacional.

Tabela 1 - Configuração do Tablet HP tx1040br

Processador	AMD Turion™ X2 Dual-Core 2.0 GHz, 1 MB L2 Cache
Memória	3072 MB DDR2 800 MHz
HD	250 GB
Resolução do visor	1280 x 800
Tamanho da tela	12.1
Tecnologias sem fios	LAN Sem Fios 802.11a/b/g/n e Bluetooth
Unidade ótica	SuperMulti DVD±RW
Webcam	WebCam HP Pavilion/ Microfone
Portas de E/S externas	3 (USB) 2.0, 1 VGA (15-pin), 1 RJ-11 (Modem), 1 RJ-45 (LAN), 1 TV-Out (S-Vídeo), 2 saídas para fones de ouvido, 1 entrada para microfone, 1 Porta de Expansão (Expansion Port 3), 2 Consumer IR (receptor de controle remoto)
Peso	1.96 kg
Dimensões do produto (L x P x A)	224 x 306 x 38 mm
Sistema operacional instalado	Windows Vista® Business

2.2. Utilização de Tablet PCs para a montagem de ambientes de ensino ativos e colaborativos

Dadas as características de hardware citadas na seção anterior, é possível entender porque o Tablet PC possibilitou e está popularizando a utilização de dispositivos computacionais na montagem de ambientes de ensino ativo e colaborativo. É fato notório que dispositivos de entrada como mouse e teclado foram elaborados de forma a valorizar a produtividade. Sendo assim, notebooks, apesar da mobilidade, tamanho de tela e poder computacional, não traziam muitos benefícios quando utilizados nesse ambiente, pois falta-lhes o principal elemento criativo: papel e caneta. Estes podem ser considerados como uma espécie de extensão de nosso cérebro, sendo essenciais para o desenvolvimento de soluções criativas [2].

Uma vez que há um dispositivo com um custo relativamente baixo, começaram a surgir os primeiros aplicativos educacionais que visavam tirar proveito desse hardware, sendo possível destacar dois deles em especial: *Classroom Presenter* [1] e *Dyknow*. Ambos têm uma base de funcionalidades semelhantes, sendo o primeiro um projeto de código fonte aberto, feito por alunos e professores da Universidade de Washington, e o segundo

(*Dyknow*) é um software proprietário que conta com mais opções de controle e segurança que o *Classroom Presenter*. A figura 3 ilustra a interface do software:

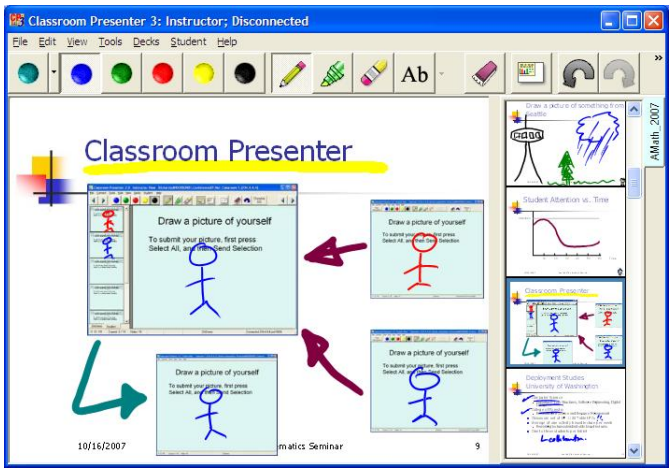


Figura 3 – Interface do Classroom Presenter

A principal vantagem oferecida pelos Tablet PCs é a capacidade de realizar anotações sobre os *slides* do instrutor. Esse fato traz benefícios normalmente ressaltados por alunos que nunca participaram de uma aula apresentada com Tablet PCs. Com a capacidade de anotar sobre seus slides, a lousa do instrutor fica mais organizada, além de poupar tempo do professor em ter que ir ao quadro explicar alguns conceitos que às vezes não podem ser feitos com a utilização de um computador dotado apenas de interfaces de entrada padrão, como mouse e teclado. A necessidade de um software próprio para apresentação é crucial, pois apesar de softwares tradicionais como o *PowerPoint* aceitarem a inserção de anotações digitais, eles não têm uma interface adequada para esse fim.

Essa vantagem apesar da relação com a montagem de ambientes de ensino ativo e colaborativo, não é suficiente para criar uma nova metodologia de ensino. Contando com a capacidade de acesso à rede *wireless* existente nos Tablet PCs, é possível trabalhar com um aplicativo onde o professor compartilhe suas transparências e anotações com outros alunos que estejam equipados com Tablet PCs, sendo que suas anotações podem ser replicadas em tempo real na tela dos alunos. Esta é outra pequena vantagem no modo de apresentação, uma vez que um aluno que tenha perdido parte do raciocínio pode revisar o conteúdo e, até mesmo, fazer anotações pessoais por cima das anotações do docente.

A próxima funcionalidade, que é o que realmente torna esse tipo de aula especial, é a introdução de áreas de escrita que podem ser compartilhadas entre todos os participantes da aula, podendo-se definir a aula como o conjunto de áreas de escrita existentes em todos os Tablet PC. Nesse contexto, há regras que permitem a organização do ambiente, a fim de organizar o fluxo de aula de forma produtiva. Podemos imaginar então que a aula passa a ter um estado global, dado pelo conjunto de áreas de escrita existentes nos Tablet PCs dos alunos e do professor. Essas áreas podem então ser compartilhadas entre os participantes, segundo alguns casos de uso. Por exemplo, um estudante resolve um exercício numa área de escrita existente em seu Tablet PC local e envia essa solução para o Tablet PC do docente, a fim de que ele a corrija.

O principal objetivo é enquadrar esses cenários de uso e adequar a utilização dessas áreas compartilhadas, de modo a atenuar alguns problemas que frequentemente ocorrem em sala de aula, dentre os quais pode-se destacar:

- O docente não possui um retorno adequado do entendimento dos alunos sobre determinado tópico. Muitas vezes, ao perguntar se a classe está entendendo determinado conteúdo, a resposta é um constragendor silêncio;
- Alunos tímidos dificilmente são participativos em sala de aula;
- Alunos com maior facilidade no aprendizado acabam atropelando alunos que necessitam de mais tempo para pensar em determinadas questões, propostas pelo docente durante a aula;
- Alunos normalmente trabalham fora da sala de aula em lista de exerícios, sendo que o docente só consegue perceber as principais deficiências da classe tardiamente (após a prova).

A proposta é que o ambiente de aula seja configurado conforme a figura 4. Ve-se assim que uma aula ativa e colaborativa com Tablet PCs intercala pedaços de aula com conteúdo expositivo e pedaços de aula em que é exigido dos alunos, de forma colaborativa, a solução de alguns problemas relacionados com o conteúdo ensinado. A solução desses exercícios é feita por grupos de pelo menos 2 alunos, sendo reservada uma área de escrita privada para cada grupo. Ao terminar a solução de um exercício, o grupo deve enviá-la para o Tablet PC do docente, de forma identificada ou anônima. O uso de Tablet PCs nesse ambiente é essencial, porque torna a montagem deste factível e otimizada, pois com papel e caneta tradicionais, a logística muitas vezes inviabiliza ambientes com tal grau de interação.

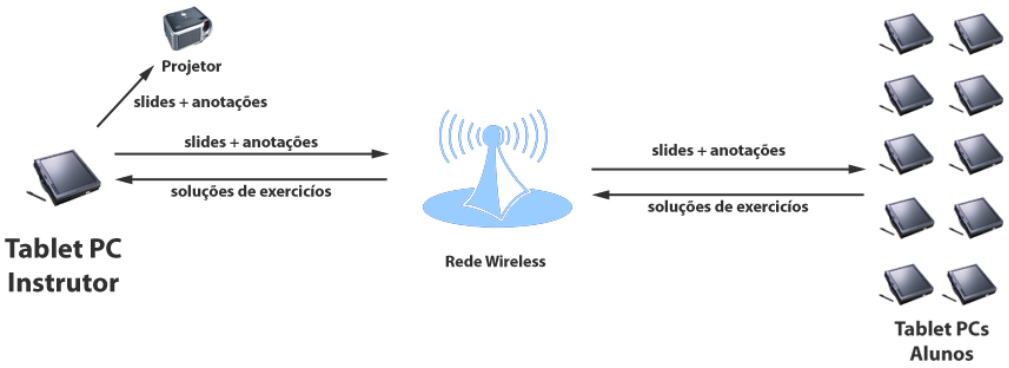


Figura 4 – Configuração de uma aula com Tablet PCs

Os problemas citados são todos abordados pois, tendo acesso às soluções, o docente pode ter uma visão clara e rápida do entendimento global, conhecendo os erros e acertos praticados por quase todos eles, resolvendo assim o problema de retorno. Diferentemente do modelo tradicional, no qual geralmente o instrutor consegue retorno somente dos melhores alunos, normalmente convidados a ir até a lousa apresentar sua solução, neste caso, é possível ter um panorama mais completo sobre o grau de entendimento da classe, otimizando o tempo de aula, focando em tópicos específicos que não foram bem entendidos pelos alunos.

Outro problema claramente abordado é a inclusão de alunos tímidos nas atividades, uma vez que a submissão da solução de seus exercícios pode ser feita de forma anônima. De maneira semelhante aos métodos tradicionais, os alunos com maior facilidade no aprendizado terminam suas soluções mais rapidamente que o restante, interrompendo assim o racícionio de estudantes mais lentos. Com o uso de Tablet PC, todos alunos têm um tempo razoável para pensar, fazendo com que todos passem a desempenhar um papel mais interessante na sala de aula. Além disso, o docente pode apresentar mais de uma forma correta de chegar ao mesmo resultado, fazendo com que os alunos exercitem sua criatividade.

O docente também consegue garantir que os alunos apliquem parte da carga horária praticando alguns exercícios. Por fim, como todo o conteúdo é digital, o docente pode disponibilizar, ao fim da aula, o conjunto de *slides* anotados para os alunos, normalmente colocando-os numa página da Internet criada para a disciplina.

Nos trabalhos mais recentes encontrados na literatura [8, 9, 10, 11 e 12] sobre a utilização desse ambiente no ensino, pode-se ressaltar a existência de diversos resultados positivos no impacto sobre os estudantes, destacando-se a preferência pela metodologia de ensino com os Tablet PCs sobre a metodologia tradicional. Há indícios de melhora quantitativa, como aumento de notas de alunos, de alunos que aprenderam utilizando essa metodologia em relação à tradicional. Os experimentos realizados até o momento, porém, foram feitos com grupos de alunos muito pequenos, não podendo assim serem tomados como verdade absoluta.

2.3. Gravação de *screencasts* utilizando Tablet PCs

Outro benefício trazido pela introdução de Tablet PCs é a possibilidade de se gravar *screencasts* das aulas. Conforme explicado anteriormente, um *screencast* consiste na gravação de um vídeo que contenha o áudio do ambiente, principalmente a voz do docente, sincronizado às anotações deste.

Na área de exatas, o conteúdo essencial de uma aula não está na expressão e gestos do docente. Apesar destes serem importantes, o material passado através dos *slides* e anotações é a essência da aula. Equipados com Tablet PCs, os docentes precisam apenas de dispositivos que permitam a captura e gravação da tela de seu *desktop* e do áudio.

Esse método, em contrapartida à gravação de aulas usando câmeras convencionais, tem a vantagem de ser extremamente barato e simples. A presença de câmeras de boa resolução e uma equipe de gravação é dispensável, uma vez que o docente pode capturar o conteúdo que antes seria exposto no quadro negro diretamente no seu Tablet PC. A taxa de captura de quadros pode ser extremamente baixa, cerca de 2 a 5 quadros por segundo, uma vez que não é necessário capturar os gestos do docente. Isso traz outro benefício indireto, que é a geração de vídeos com uma baixa taxa de bits. Os vídeos ficam com um tamanho final diminuto se comparados aos vídeos de aulas capturados com câmeras, tendo assim requerimentos de banda de transmissão extremamente relaxados, fazendo com que tanto o seu upload para um servidor, quanto a visualização pelos alunos seja muito rápida.

Um software que é comumente utilizado na geração de *screencasts* é o *Camtasia Studio*. Ele permite que o docente grave sua tela e seu áudio, podendo gerar, ao final da aula, um arquivo de vídeo no formato escolhido. O *Camtasia* suporta a maioria de *codecs*

populares como *flash video* e *windows media video*, sendo escolha do docente o formato mais adequado. Uma vez gravados, o instrutor pode disponibilizar esses *screencasts* de diversas formas, sendo que a mais indicada é colocá-los junto aos *slides* anotados na página da Internet da disciplina.

Atualmente, é possível encontrar na literatura, como proposto por Carryer [4], diversos usos para os *screencasts*, indo de abordagens mais convencionais, onde o vídeo é disponibilizado apenas como material extra de consulta a abordagens mais agressivas, onde os cursos são modificados de forma a terem uma metodologia de ensino híbrida, na qual os alunos devem assistir os vídeos previamente às aulas, sendo estas reservadas para a execução de exercícios ou trabalhos práticos. Ambas abordagens têm se mostrado bem sucedidadas, mas no caso deste trabalho, a abordagem utilizada visa somente a geração de material de consulta extra, que pode ser utilizado pelo alunos para o esclarecimento de dúvidas pontuais ou, ainda, consultado em caso de faltas.

A captura de vídeos, dependendo da resolução utilizada, da área de tela capturada, da qualidade de som e de alguns outros fatores, pode precisar de um parcela significativa de poder de processamento do computador, sendo recomendado, então, a utilização de Tablet PCs com uma boa quantidade de memória, juntamente com um processador *dual-core* (um dos *cores* fica responsável pela codificação do vídeo e o outro pelas funções do software de apresentação) e um disco rápido.

3. Metodologia

3.1. Infra-estrutura

Desde 2002, a UNICAMP conta com um laboratório equipado com Tablet PCs. Nesse ano, três Tablet PCs modelo HP TC1100 foram doados ao instituto pela *Microsoft Research*. Nesse período inicial não havia cenário de uso, sendo que muitos professores ainda ministravam seus cursos utilizando transparências físicas, anotando por cima destas quando necessário. No primeiro semestre de 2004 foi ministrado um curso de Arquitetura de Computadores utilizando o Tablet PC, sendo o Tablet PC, operado apenas pelo docente. Seu uso consistiu basicamente numa plataforma para projeção de transparências digitais e anotação do docente em cima das mesmas, não existindo, nesse período, softwares capazes de armazenar as anotações do docente. Este somente poderia efetuá-las num slide e, ao trocar para outro, as mesmas desapareciam. Outro problema era a impossibilidade de criar um novo slide em branco. Esse fato, apesar de parecer simples, atrapalha toda a logística da aula, uma vez que, caso o docente queira escrever algo num slide em branco, ele terá que ir até o quadro negro, provocando uma quebra no ritmo da aula e perda da capacidade de digitalizar automaticamente o conteúdo escrito. Outro fator a atrapalhar esta dualidade na aula é a luminosidade da frente da sala, que tem que ser reduzida para a projeção e ampliada para melhor visualização do quadro.

Ainda assim, essa forma de apresentação, com transparências digitais, deu maior liberdade e qualidade ao conteúdo dos *slides* utilizados pelo docente. Na área de Arquitetura de Computadores esse fato é extremamente importante, visto que boa parte do conteúdo do curso são esquemas de processadores, incluindo máquinas de estados, *datapaths*, *pipelines* e esquemas de barramentos. A capacidade do docente de escrever por cima desses conteúdos foi motivo de grande sucesso entre os alunos, tornando a aula

mais organizada e com uma grande riqueza visual na explicação dos detalhes da disciplina.

No começo de 2007, a UNICAMP recebeu uma doação de 21 Tablet PCs da HP, modelos TC4200 e TC4400, mais hardware para a montagem de uma rede *wireless*, fruto do envio de uma proposta entitulada "*An annotation-based tool for collaborative learning using mobile technology"* a um *Technology for Teaching Grant* da HP. Assim, foi possível montar um laboratório para a prática de ensino ativo e colaborativo descrito nos moldes da seção anterior. Esse foi o material usado durante as aulas do curso avaliado nesse capítulo.

3.2. Descrição do curso

O curso de Arquitetura de Computadores da UNICAMP tem a duração de um semestre e é oferecido apenas uma vez por ano, ministrado, desde 2004, pelo prof. Rodolfo Jardim de Azevedo. O curso segue o tradicional livro texto de David A. Patterson and John L. Hennessy, "*Computer Organization Design, The Hardware/Software Interface"* [6].

Nos anos anteriores, a não ser pela metodologia de apresentação diferenciada pelo uso de Tablet PCs, a estrutura do curso foi tradicional, contando com aulas magnas, listas de exercícios e trabalhos sobre temas pré-definidos. Em 2008 o curso manteve estrutura semelhante aos anos anteriores, porém o docente passou a gravar os *screencasts* de suas aulas utilizando o *Camtasia Studio*. Em partes do curso, principalmente aquelas em que a explicação do conteúdo envolvia grande utilização de diagramas e figuras, foram distribuídos Tablet PCs aos alunos e problemas relacionados ao tema eram propostos, para resolução e envio de soluções para ao professor durante a aula.

A tabela 2 resume o calendário de aulas, mostrando em quais destas foram gravados vídeos ou realizados exercícios ativos e colaborativos. Para cada linha da tabela é indicado o conteúdo ministrado e a data da aula, bem como é marcado um X para ilustrar que foi gravado um vídeo dessa aula ou que foram realizadas atividades ativas e colaborativas. Na primeira linha da tabela, por exemplo, podemos ver que no dia 06/08, uma quarta-feira, foi dada a aula de Apresentação do Curso e Introdução, e para essa aula foi gravado o vídeo mas não houve a realização de exercícios ativos e colaborativos. É importante ressaltar que era desejado introduzir a nova tecnologia passo a passo, a fim de permitir que os estudantes se acostumassem com esta. Primeiro, os estudantes tiveram a oportunidade de entrar em contato com o Tablet, através da nova forma de apresentação e da gravação dos *screencasts*, deixando para começar a usar o Tablet PC efetivamente após algumas aulas.

Tabela 2 - Calendário do curso

Data	Conteúdo	Vídeo	Exercício com Tablet PC
06/08Q	Apresentação do Curso – Introdução	X	
11/08S	Capítulo 2 – Cojunto de Instruções	X	

13/08Q	Capítulo 2 - Conjunto de instruções	X	
18/08S	Capítulo 2 - Conjunto de instruções	X	
20/08Q	Capítulo 3 - Multiplicador e Divisor	X	
25/08S	Capítulo 3 - Representação de Ponto Flutuante	X	
27/08Q	Capítulo 4 – Desempenho	X	
01/09S	Capítulo 4 - Desempenho Apêndice B - Construção da ALU	X	
03/09Q	Capítulo 5 - Construção do Datapath monociclo	X	
08/09S	Capítulo 5 - Datapath monociclo e controle	X	
10/09Q	Capítulo 5 - Datapath monociclo e controle	X	
15/09S	Controle do datapath monociclo	X	X
17/09Q	Microprogramação	X	X
22/09S	Capítulo 6 – Pipeline	X	X
29/09S	Capítulo 6 – Pipeline		
01/10Q	Capítulo 6 – Hazards	X	X
06/10S	Capítulo 6 - Branch Prediction	X	X
08/10Q	Aula de Exercício	X	
13/10S	Prova 1		
20/10S	Capítulo 7 – Cache	X	X
22/10Q	Capítulo 7 – Cache		X
03/11S	Capítulo 7 - Memória Virtual	X	X
05/11Q	Capítulo 8 – Entrada e Saída	X	X
10/11S	Capítulo 8 – Entrada e Saída	X	X
12/11Q	Multicore		
17/11S	Aula de Exercício		
19/11Q	Prova 2		
24/11S	Apresentações de Trabalhos		
26/11Q	Apresentações de Trabalhos		
10/12Q	Exame Final		

Os *slides* de todas as aulas forma disponibilizadas de antemão para os alunos. Aquelas anotadas pelo professor, as soluções de exercícios em sala de aula e as submissões dos alunos foram disponibilizadas no site da disciplina, para que os alunos pudessem acessá-las após a aula correspondente. A figura 5 ilustra exemplos de transparências anotadas pelo docente durante a aula, que foram posteriormente dispobilizadas para os alunos. Novamente ressaltamos que, devido ao uso do Tablet PC, estas anotações foram feitas exatamente sobre o conteúdo projetado em sala de aula. Os *screencasts* gravados foram disponibilizados no máximo 1 semana após a aula ministrada, sendo também postados como um link na página de disciplinas.

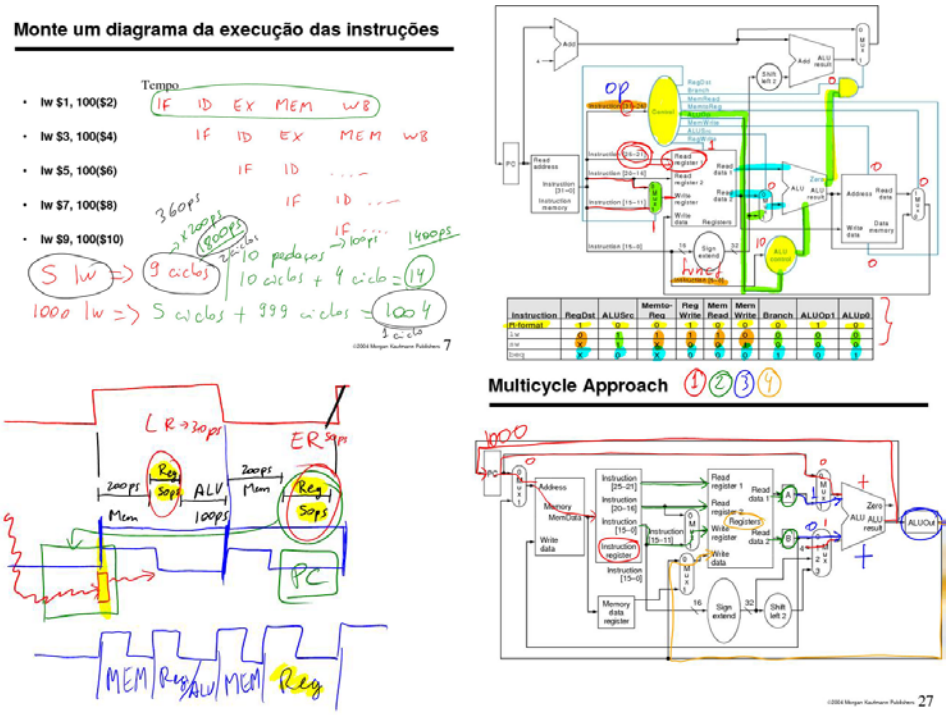


Figura 5 – *Slides* anotados pelo docente

O curso contou com um total de 79 alunos matriculados em seu início. Destes, 1 aluno trancou a disciplina, 5 foram reprovados por falta e 6 foram reprovados por nota. Esses dados serão comparados com os cursos de semestres anteriores, na seção de resultados.

Nas aulas colaborativas, os alunos foram instruídos a formarem grupos de 4 pessoas, pegarem um Tablet PC no início da aula e seguirem um breve procedimento de inicialização e conexão com o Tablet PC do instrutor. Esse processo levou cerca de 5 a 10 minutos para ser executado por aula, sendo um pouco mais demorado na primeira. Alguns alunos optaram por seguir a aula sem a utilização do dispositivo durante as aulas ativas e colaborativas - no entanto foram minoria e, em alguns momentos do curso foram convidados a testarem o equipamento, não havendo, porém, insistência por parte do docente ou obrigatoriedade na utilização do equipamento. Todos os Tablet PCs utilizaram o sistema operacional Windows Vista e o software Classroom Presenter para se conectar ao Tablet PC do docente. A figura 6 ilustra algumas soluções dos alunos que foram submetidas para o Tablet PC do docente.

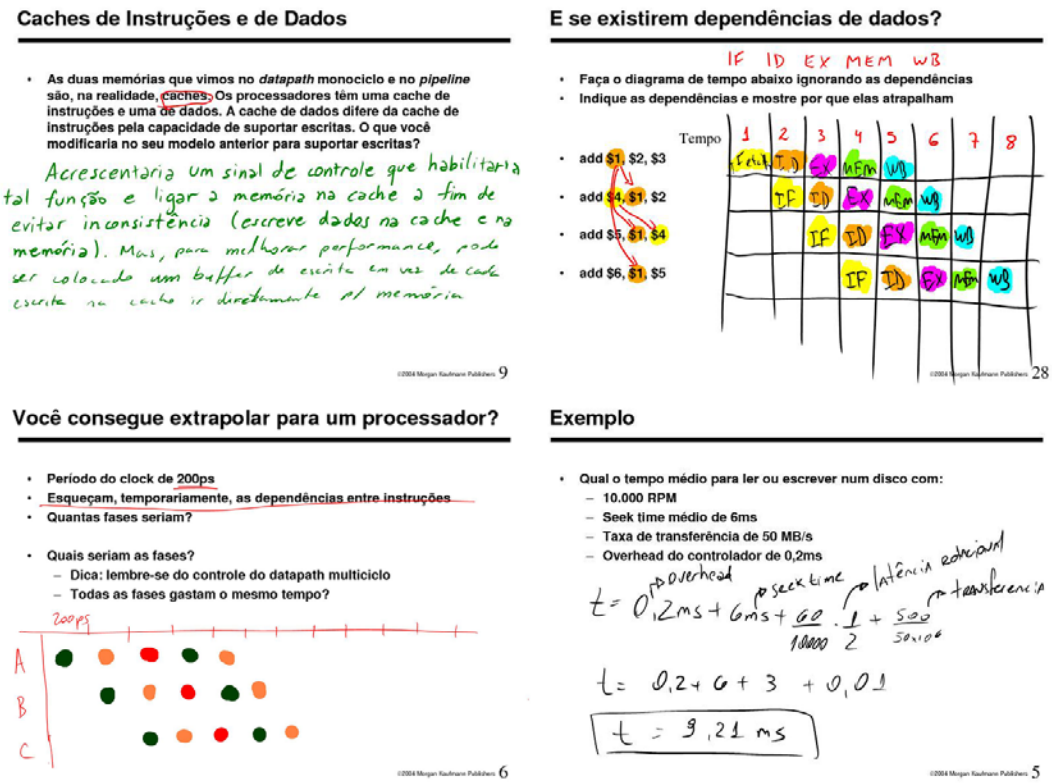


Figura 6 – Exemplos de exercícios enviados pelos alunos

Dadas essas mudanças no curso, foi necessário elaborar um método de avaliação que permitisse uma análise de impacto destes no aprendizado dos alunos, bem como avaliar, do lado do docente, se o retorno dado pelos alunos durante as aulas o ajuda a ter uma ideia mais precisa do entendimento destes, sobre o conteúdo ensinado.

Para medição do impacto dos alunos foram utilizados questionários, baseados em outros que já vinham sendo utilizados em trabalhos anteriores, adaptados às necessidades da avaliação nessa disciplina. O principal objetivo desses questionários era obter informações quantitativas e qualitativas sobre a metodologia de apresentação do curso, o grau de impacto das aulas ativas e colaborativas no aprendizado dos alunos, os hábitos de estudo fora da classe e a aceitação do Tablet PC por parte dos alunos.

4. Resultados

Ao fim do semestre, foram aplicados os questionários referidos na seção anterior. Apesar do custo do equipamento não ser proibitivo, sabe-se que nem toda instituição de ensino pode dispor de verba suficiente para montar um laboratório que suporte a realização de aulas ativas e colaborativas, conforme descrito neste capítulo. Para tanto, a análise de resultados está dividida em quatro subseções. As duas primeiras irão analisar os parâmetros pertinentes à utilização de Tablet PCs nas aulas ativas e colaborativas, sendo a primeira referente à avaliação feita pelos alunos e a segunda a avaliação feita pelo docente, permitindo assim que o leitor tenha uma clara visão dos benefícios e desvantagens de se montar um laboratório ou sala de aula com Tablet PCs. As duas últimas subseções, a exemplo da primeira avaliação, analisam os parâmetros pertinentes

à utilização do Tablet PC como plataforma de apresentação e de *screencasts*, sob o ponto de vista dos alunos e do docente. As mudanças relativas à apresentação e gravação de *screencasts* demandam somente um Tablet PC (para o docente) sendo muito mais atrativas do ponto de vista financeiro e logístico.

Foram passíveis de análise medidas que indiquem mudanças nos hábitos, no comportamento e na motivação dos alunos durante o curso. Não foram levadas em conta medidas como notas finais, pois não foi montado um experimento adequado para tanto. Apesar do conteúdo ser o mesmo, os cursos são dinâmicos e a dificuldade das provas pode variar baseada na percepção do entendimento da classe pelo professor, bem como o ritmo da matéria ministrada durante o curso. Isso faz com que comparações de rendimento entre alunos desse semestre e de semestres distintos, embora válidas do ponto de vista informal para verificação de algo extraordinário, sejam impróprias para um estudo científico. Outros parâmetros, no entanto, como porcentagens de aprovação e reprovação e número de faltas, por sua maior regularidade, podem nos mostrar algo mais significativo sobre a mudança de comportamento dos alunos.

4.1. Resultados do uso do Tablet PC em aulas ativas e colaborativas: Visão dos alunos

Para avaliar como os alunos reagiram à nova metodologia das aulas, foi aplicado um questionário ao final do curso. Dos 79 alunos matriculados na disciplina, 49 responderam a esse questionário. As respostas foram organizadas segundo a escala de Likert, com 5 gradações, sendo o texto ajustado quando necessário.

O primeiro parâmetro medido, relativo ao uso de Tablet PCs em aulas ativas e colaborativas, é a dificuldade de adaptação dos alunos a esse novo ambiente. A figura 7 mostra os resultados obtidos:

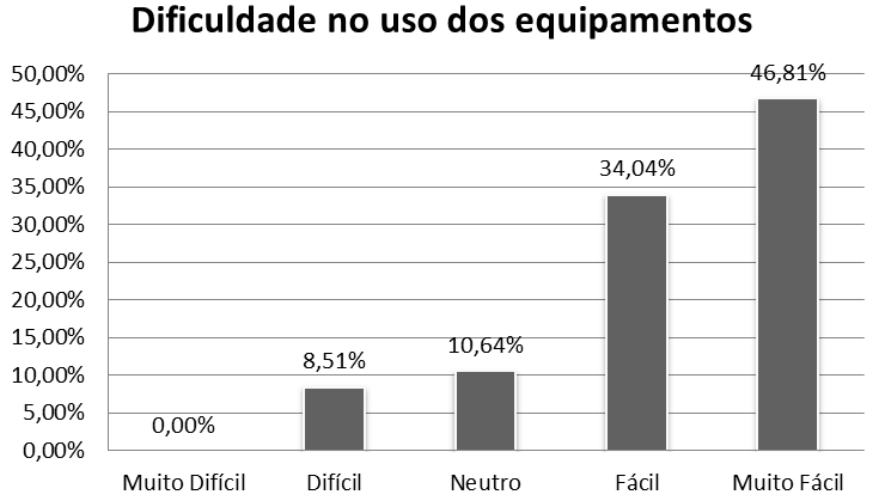


Figura 7 – Dificuldade de adaptação dos alunos ao ambiente ativo e colaborativo

O gráfico nos mostra que a grande maioria dos alunos (81%) disse que se adaptou de forma muito fácil ou fácil ao novo ambiente. Na primeira aula ativa e colaborativa são

dadas breves instruções de utilização, que mostraram-se suficientes para que os alunos entendessem o funcionamento do equipamento e a dinâmica da aula. Poucos alunos tiveram dificuldade, sendo ajudados pessoalmente, quando necessário.

O segundo parâmetro em questão refere-se à atenção dos alunos na sala de aula. A figura 8 ilustra o resultado obtido:

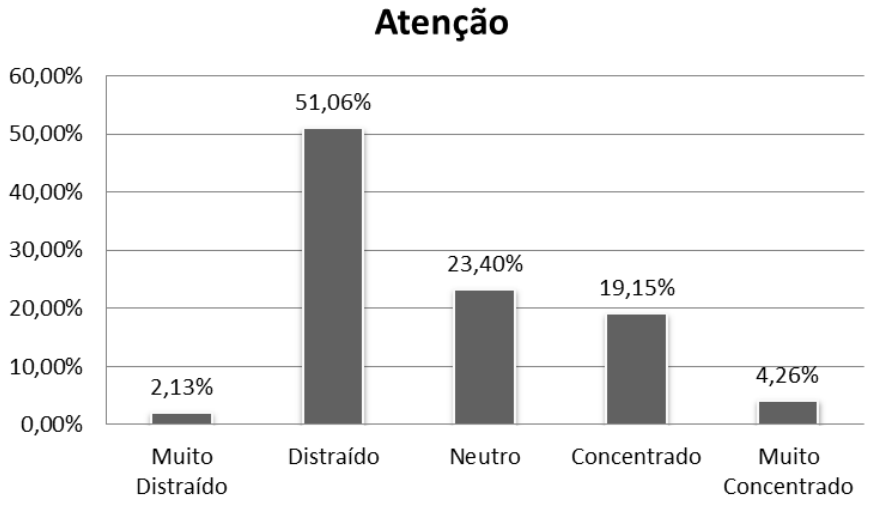


Figura 8 – Atenção dos alunos durante as aulas ativas e colaborativas

Uma porcentagem alarmante dos alunos, 51%, disse ter se sentido mais distraída nessas aulas, em comparação a aulas tradicionais. Acredita-se que a relação alunos por Tablet PC, especialmente elevada nessa classe, de 1 Tablet PC para cada 4 alunos é a principal causa de tal distração. Os Tablet PCs foram utilizados numa sala de aula tradicional, com cadeiras e mesas individuais, dispostas em linhas e colunas. Há uma sala especial na Unicamp, equipada com mesas e cadeiras que permitem uma melhor disposição dos alunos, permitindo assim uma melhor interação entre os grupos de alunos. Nessa sala, no entanto, os alunos que se sentavam mais ao canto tinham alguma dificuldade para trabalhar com seus colegas. Alguns comentários dos alunos ilustram que estes tiveram uma visão semelhante do problema:

"Mais Tablets para diminuir a divisão do tempo de uso entre alunos, e cadeiras para melhor apoiar (ou mesas) o Tablet."

"Acho que um maior número de Tablets disponíveis sanaria o problema da distração causada."

"Sala grande atrapalhou o uso do Tablet PC, pois ficaram muito espaçados os alunos."

"As aulas com o Tablet PC foram muito boas, apesar de facilitar a distração. Poderia haver um aumento no número de Tablet PCs."

Outros fatores que podem ter contribuído na distração dos alunos foram a complexidade dos exercícios aplicados e a acústica da sala. Alunos frisaram que o uso de exercícios pontuais, para fixação de conteúdo, seria mais eficientes e ajudaria a resolver parte do problema da distração, já que os eles não demorariam muito tempo para resolvê-los. Os seguintes comentários deixam o problema mais claro:

"Acredito que o principal motivo da dispersão em sala de aula após o início da utilização do Tablet PC foi o tamanho da sala e a acústica ruim para uma turma dividida em grupos que fatalmente vão discutir durante a aula."

"A acústica das salas do PB é horrenda, sem contar a qualidade dos assentos."

"Um dos pontos que me tiravam a atenção da aula era ver o que pessoas na minha frente estavam fazendo no Tablet."

Apesar do problema com a distração, houve um comportamento muito interessante por parte dos alunos. Os estudantes que acabavam os exercícios mais rapidamente começaram a ajudar outros grupos a resolverem seu exercício, fazendo com que a classe trabalhasse de forma mais unida.

O problema das mesas e cadeiras não adequadas já era esperado, sendo incluída uma questão no questionário sobre o conforto dos alunos em sala. A falta de apoio adequada, bem como o pouco espaço existente entre uma cadeira e outra, poderiam atrapalhar os alunos durante as aulas ativas com Tablet PC. Apesar das expectativas negativas em relação ao conforto, a figura 9 mostra que a maioria (74%) dos alunos se sentiu confortável ou muito confortável na sala de aula.

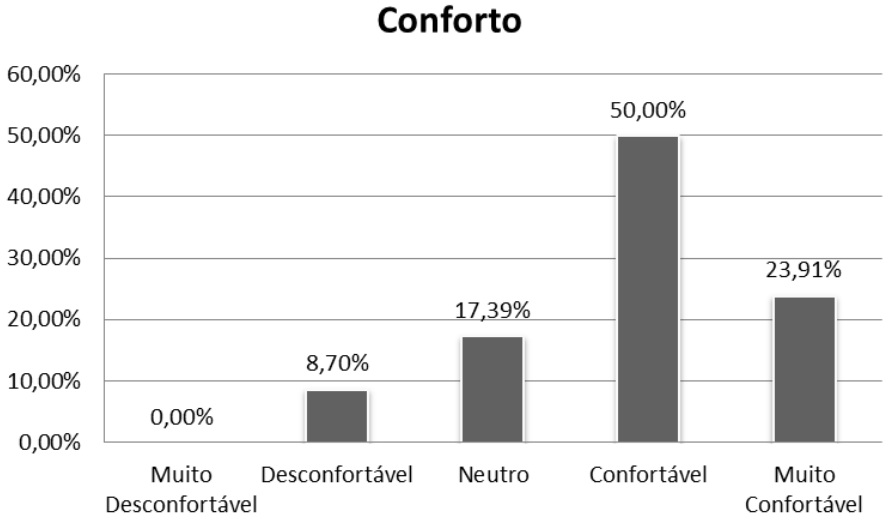


Figura 9 – Conforto dos alunos em sala de aula

Apesar dos alunos terem se sentido mais dispersos durante a aula, o questionário mostrou dois dados importantes: os alunos acharam que sua participação, tanto individualmente dentro do grupo, quanto individualmente dentro da sala de aula, aumentou. A figura 10 mostra os gráficos que indicam essa mudança:

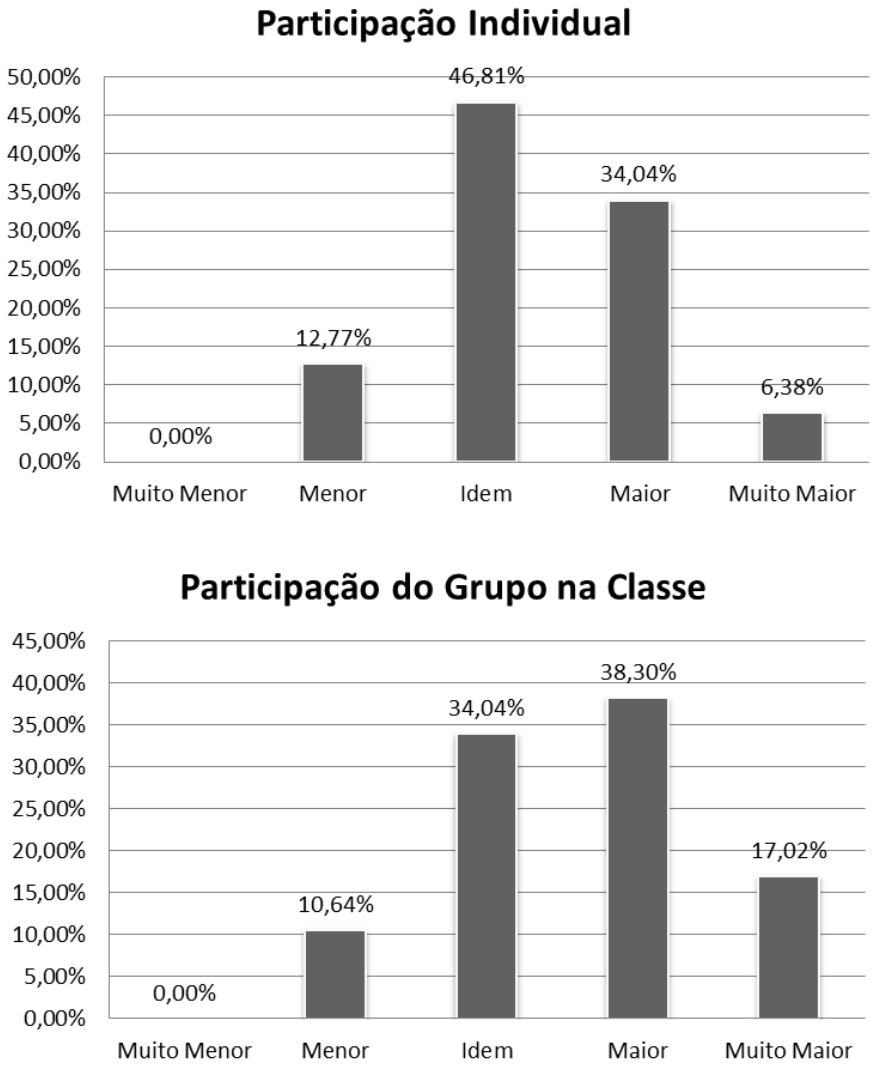


Figura 10 – Participação dos alunos

Pode-se notar que um dos objetivos iniciais, o de tornar os alunos mais participativos em sala de aula, foi atingido. O ambiente ativo e colaborativo, montado dessa forma, permite que os alunos tímidos e aqueles que precisam de um pouco mais tempo de raciocínio do que os melhores alunos da classe também possam participar ativamente das aulas com Tablet PC. Nesse ponto, vale ressaltar que a submissão dos exercícios pode ser feita de forma anônima e esses alunos, antes excluídos, podem ter sua solução escolhida pelo professor para ser alvo de uma discussão com a classe. Outro fato relevante é que a diversidade de soluções com que o professor tem contato é muito maior, permitindo que ele tenha uma visão geral do entendimento da classe e que ele visualize erros comuns cometidos pelos alunos, corrigindo-os imediatamente. Além disso, é possível mostrar para a classe mais de uma solução, valorizando alunos

criativos e fazendo com que a turma tome contato com diferentes soluções, aumentando seu senso crítico.

A comprovação da melhor participação do professor está indicada na figura 11, que mostra a percepção dos alunos sobre o acesso ao professor, para esclarecimento de dúvidas durante a aula:

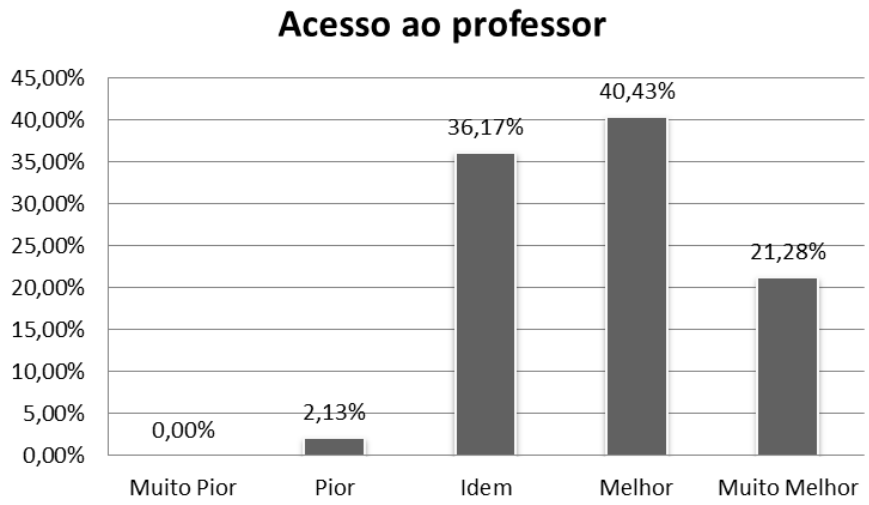


Figura 11 – Acesso ao professor para esclarecimentos de dúvidas durante a aula

Diretamente do gráfico, é possível notar que maioria dos alunos, 62%, achou que as aulas ativas e colaborativas permitem um acesso ao professor melhor ou muito melhor, sendo que apenas uma parcela muito pequena (2%) achou a nova abordagem pior em relação ao acesso ao professor. Conclui-se, então, que o uso de Tablet PCs em aulas ativas e colaborativas permite ao docente se comunicar melhor com os alunos, fazendo com que os mesmos notem que o instrutor consegue esclarecer melhor as dúvidas levantadas pela classe.

Outro ponto a ser avaliado pelos alunos é verificar como a nova metodologia de aplicação de exercícios se compara ao modelo tradicional. É comum o instrutor passar um exercício para os alunos no quadro negro, que devem então resolvê-lo numa folha de papel. Por fim, um aluno escolhido para apresentar sua solução na lousa. Assim, foi perguntado aos alunos se papel e caneta eram preferíveis ao Tablet PC. A ordem da pergunta nesse caso é importante, uma vez que os participantes de um questionário têm certa tendência em concordar com a afirmação que lhes é proposta. Como o interesse é validar essa nova metodologia, inverteu-se a ordem lógica da pergunta, pois assim garante-se que, se houver alguma tendência, ela será direcionada para a preferência de papel e caneta. A figura 12 mostra os resultados obtidos para essa pergunta:

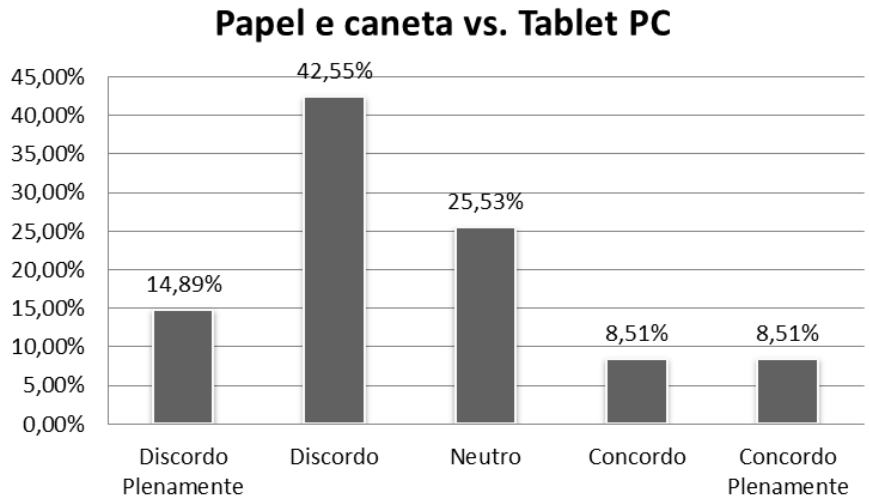


Figura 12 – Preferência dos alunos: papel e caneta vs. Tablet PC

A maioria dos alunos disse discordar da afirmação, mostrando que preferem a utilização de Tablet PC para a realização dos exerícios, ao invés de papel e caneta. Além disso, alguns alunos citaram que o fato do Tablet PC permitir que eles visualizassem qualquer transparência do docente e suas anotações, fez com que eles pudessem acompanhar melhor a aula. Uma descrição emblemática disso é:

"O recurso de poder voltar em slide e relê-lo é muito poderoso. Dependendo da disciplina se você perdeu um slide você perdeu o resto da aula."

Por fim, a figura 13 mostra os resultados encontrados, quando os alunos foram perguntados sobre a importância do Tablet PC no seu aprendizado nessa disciplina.

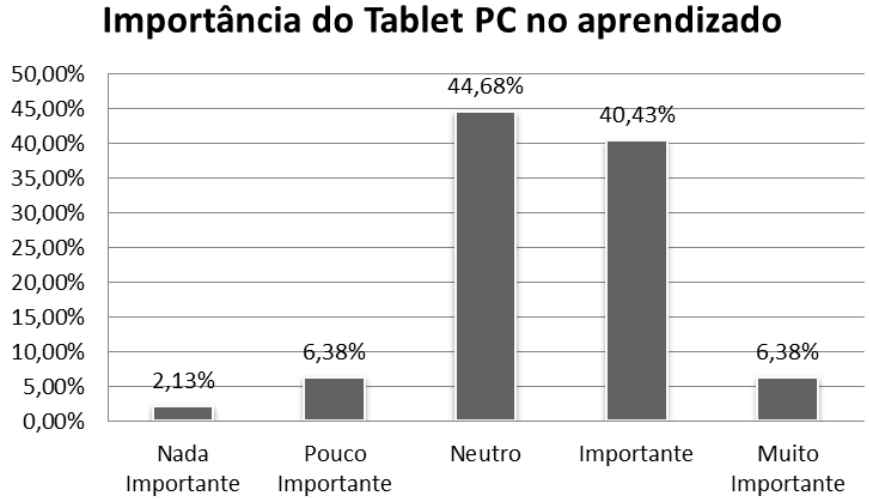


Figura 13 – Importância do Tablet PC no aprendizado dos alunos na disciplina de Arquitetura de Computadores

Uma parcela muito pequena achou que o Tablet PC não colaborou em nada com seu aprendizado. Uma pacela significativa achou que o Tablet PC não teve influência no seu processo de aprendizado, adotando uma posição de neutralidade. Outra parcela significativa ressaltou a importância do Tablet PC no seu aprendizado. Nesse ponto pode-se destacar que quase metade dos alunos encontrou alguma importância no Tablet PC durante seu processo de aprendizado (47%).

4.2. Resultados do uso do Tablet PC em aulas ativas e colaborativas: Visão do docente

A possibilidade de fornecer atividades durante a aula e ter as respostas dos alunos disponíveis instantaneamente, para avaliar o nível de retenção, cria novas alternativas de ensino. A partir da introdução do Tablet PC em sala, foram incluídas atividades de motivação no início de cada aula, onde os alunos, em grupos, deviam resolver pequenas tarefas relacionadas com o tema da aula. Antes da aula sobre *caches*, os alunos foram expostos a uma listagem de números representando endereços de memória, com a restrição de que teriam apenas 4 lugares temporários para armazenar os números mais utilizados. Na primeira versão, eles sabiam todos os números que seriam apresentados e indicaram os números que mais ocorriam. Em seguida, eles deviam realizar a mesma operação, mas sem levar em conta o conhecimento sobre a listagem, tomando decisões passo a passo. A terceira atividade consistia em modelar uma forma de endereçar rapidamente os 4 lugares temporários, conforme o endereço de entrada. Para cada um destes passos, as soluções intermediárias de vários grupos de alunos eram comentadas, anonimamente, na frente da sala, e as alternativas mais factíveis eram escolhidas para dar prosseguimento à aula.

4.3. Resultados do uso do Tablet PC para apresentação da lousa e para gravação de *screencasts*: visão dos alunos

Esta seção apresenta como o comportamento dos alunos foi afetado pelo fato de terem acesso aos *screencasts* das aulas, e qual sua reação ao novo modo de apresentação. Quanto ao modo de apresentação, nenhum aluno apresentou queixa, pelo contrário, os alunos acharam que as aulas foram bastante claras e ricas visualmente. Sua fácil adaptação ao ambiente, conforme discutido anteriormente, mostra que os alunos não tiveram problema algum quanto ao uso do Tablet PC associado ao projetor. Não surgiram reclamações relativas ao ritmo das aulas, o que era recorrente em semestres anteriores.

Elaborou-se um questionário para avaliar o comportamento dos alunos com as mudanças introduzidas. Particularmente, o interesse era verificar como a distribuição dos *slides* anotados pelo professor e dos *screencasts* modificava a forma de estudo dos alunos. Além disso, estatísticas de utilização, como a quantidade de alunos que acessou os *screencasts* como material de estudo, a quantidade de *screencasts* vistos e as partes mais assistidas também fizeram parte do questionário. Eles também foram questionados sobre sua assiduidade à sala de aula frente a existência dos vídeos, interesse em assistir as aulas remotamente, importância de indexação, a fim de facilitar a visualização de trechos específicos das aulas (nesse caso não havia indexação, conforme será explicado posteriormente). Por fim, questionou-se a facilidade para assistir os vídeos, os problemas ocorridos ao tentar faze-lo e, ainda, sobre a qualidade destes. Esse

questionário foi respondido por 58 dos 79 alunos matriculados no curso. Todos os resultados aqui apresentados são referentes a esses 58 alunos.

A figura 14 mostra que a grande maioria dos alunos (dos 58 que responderam o questionário) utilizou os vídeos, sendo 86% dos estudantes. Os estudantes que não utilizaram o vídeo não apresentaram os motivos para tanto. Um levantamento no servidor apontou uma média de 36 acessos únicos para cada *screencast*, sendo o número mínimo de acessos igual a 18 e o máximo igual a 50. Dezessete alunos estimaram a quantidade de horas de vídeo assistidas, sendo que o resultado uma média de 18 horas de vídeo. Quatorze alunos estimaram o número de vídeos que foram consultados por eles, chegando a uma média de 11 vídeos.

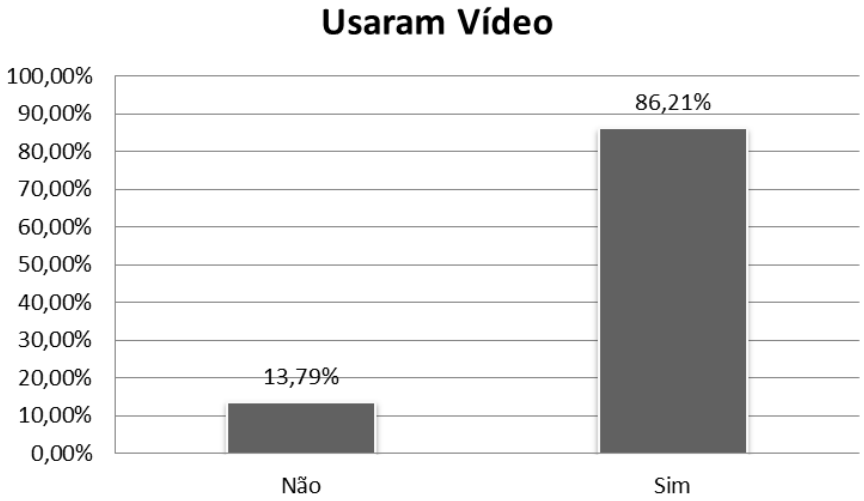


Figura 14 – Percentagem de alunos que utilizaram vídeos como material de estudo

Alguns comentários dos alunos que ilustram o uso dos vídeos:

"Qtde de horas: pelo menos 5 horas, foi mais utilizado p/ tirar dúvidas de coisas que não entendi ou alguma coisa extra ao livro."

"Aproximadamente 5 vídeos na 1ª prova e 1:30hs de vídeos diversos na 2ª prova."

"Os vídeos (as aulas) abrangem bastante o conteúdo proposto no livro-texto. É uma ótima idéia a ser continuada, mas talvez a qualidade do som poderia melhorar. Tirando as aulas de introdução eu assisti todos os vídeos."

Também foi perguntado para os alunos se estes tiveram problemas na utilização dos vídeos. O formato escolhido para publicação foi o *Flash Video*: uma vez que sites populares como o YouTube utilizam esse formato é esperado que a grande maioria dos usuários tenha o software necessário para assistir os vídeos instalados em suas máquinas. A figura 15 mostra que uma quantidade pequena, embora não desprezível, dos alunos (20%) teve problema na visualização dos vídeos. Era pedido aos alunos que tiveram problemas para que detalhassem o ocorrido. A maioria das reclamações se deveu ao fato do servidor de vídeos não estar disponível em alguns momentos. Os vídeos foram colocados numa máquina localizada na sala do docente, junto com o site

da disciplina. Eventualmente, problemas como queda de rede no Instituto de Computação, ou desligamentos de energia para reparos na rede elétrica do campus da Unicamp, foram os principais responsáveis por esse problema. Apenas um aluno reclamou que uma máquina do laboratório de informática do instituto estava sem o software necessário para visualização dos vídeos.

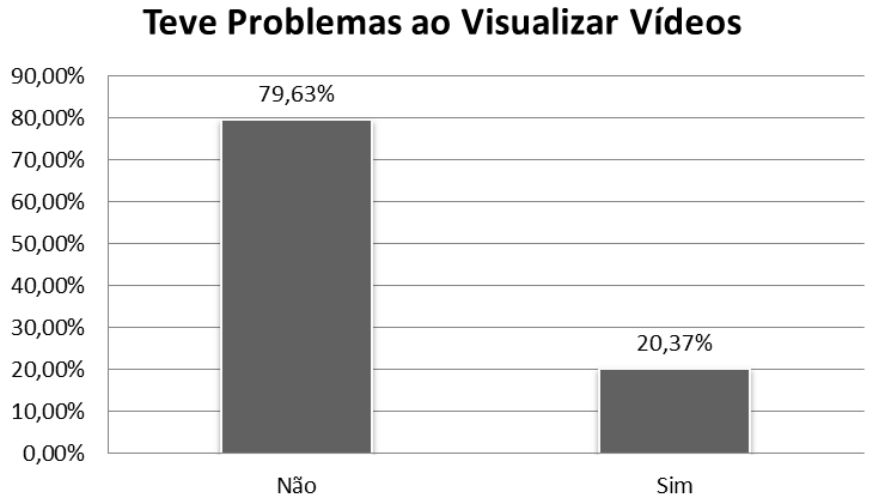


Figura 15 – Percentagem de alunos que tiveram problemas na visualização dos vídeos

Alguns comentários que ilustram os problemas enfrentados pelos alunos:

"Em alguns momentos os vídeos não abriam, mas eram raros."

"Alguns não carregavam."

"Conexão lenta atrapalha. Seria interessante poder baixar."

Como os vídeos aqui referidos são uma reprodução da tela projetada pelo instrutor sincronizada com seu áudio, foi perguntado aos alunos se a qualidade dos vídeos era boa o suficiente para ser assistida posteriormente. A figura 16 mostra que nenhum aluno achou os vídeos ruins ou muito ruins, com praticamente todos alunos (94%) dizendo que a qualidade destes foi boa ou muito boa. Em matérias onde há uma grande densidade de figuras, símbolos, diagramas e esquemas, seria muito oneroso capturar a aula. Esses materais são a essência da aula, sendo que para capturá-los num vídeo convencional seria necessário pelo menos uma câmera e um operador, além do vídeo ter que ser disponibilizado numa alta resolução para legibilidade dos detalhes. O uso do Tablet PC permite que o docente, sem prejuízo do conteúdo da aula, possa gravá-la apertando um simples botão, além de gerar um vídeo com um tamanho muito reduzido (cerca de 60MB/hora) e boa resolução. Mesmo com esse tamanho, alguns alunos com conexões de Internet mais lentas reclamaram da velocidade, sendo esse um fator de importância na qualidade dos vídeos. Outro ponto citado foi a qualidade dos vídeos e o fato de perguntas dos alunos não serem capturadas pelo microfone do docente. Alguns comentários dos alunos que ilustram esses problemas foram:

"O áudio do vídeo deixou um pouco a desejar. Não foi possível ouvir perguntas dos alunos."

"Não é possível escutar o que os alunos falam (suas dúvidas e respostas)."

"Editar vídeos removendo dúvidas inaudíveis e piadas fora de contexto."

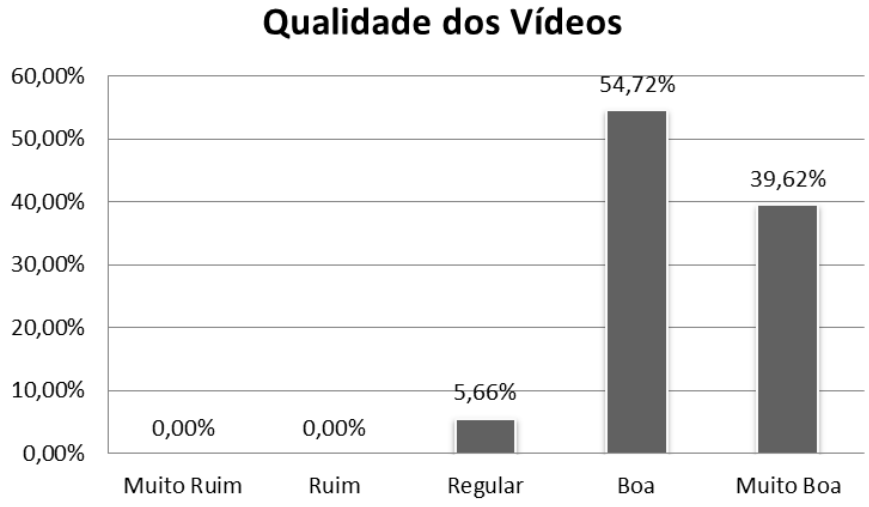


Figura 16 – Qaulidade dos vídeos segundo os alunos

Uma vez que a principal intenção ao gravar os vídeos é fornecer aos alunos um material extra para estudo, uma variável importante a ser medida é a introdução de alguma forma de indexação dos vídeos, permitindo que os alunos naveguem mais facilmente para o trecho desejado. A figura 17 ilustra que 81% dos alunos acharam que é importante ter alguma forma de indexação, mostrando assim que agrande maioria realmente não está interessada em utilizar o vídeo como substituto da aula e, sim consultar partes deste para esclarecimento de dúvidas.

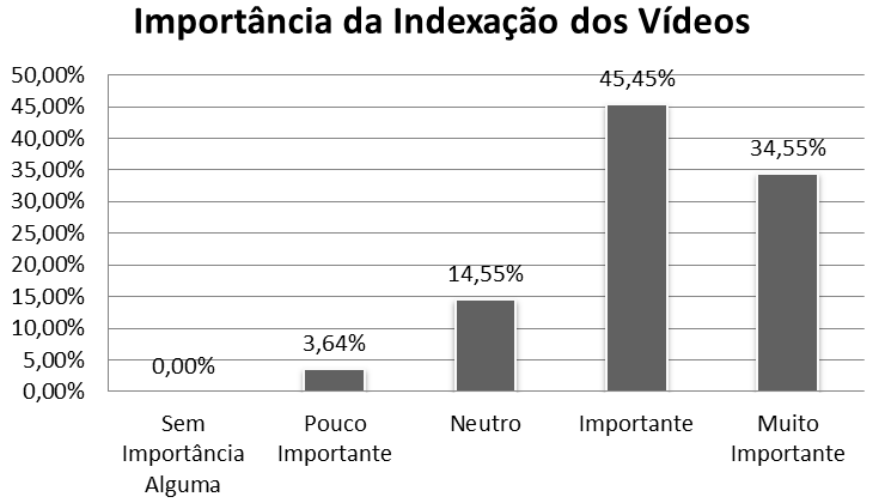


Figura 17 – Importância da indexação dos vídeos (para busca)

Apesar dessa tendência de utilização de partes do vídeo para revisão, alguns alunos utilizaram o vídeo como substituto da aula toda em alguns casos especiais. A figura 18 mostra que 26% dos alunos que tinham que faltar a uma aula preferiram fazê-lo nessa matéria. Alunos que tinham compromissos pessoais, como consultas médicas e exame de carteira de motorista, deixaram para realizar tais atividades no horário da disciplina de Arquitetura de Computadores, porque sabiam que teriam os vídeos e *slides* do professor disponíveis para consulta via WEB. Os alunos foram questionados sobre quantas faltas eles tiveram, graças à disponibilização dos vídeos. Foi encontrada uma média de 3 faltas, sendo o máximo 6, e o máximo de faltas permitidas eram 8 durante o semestre.

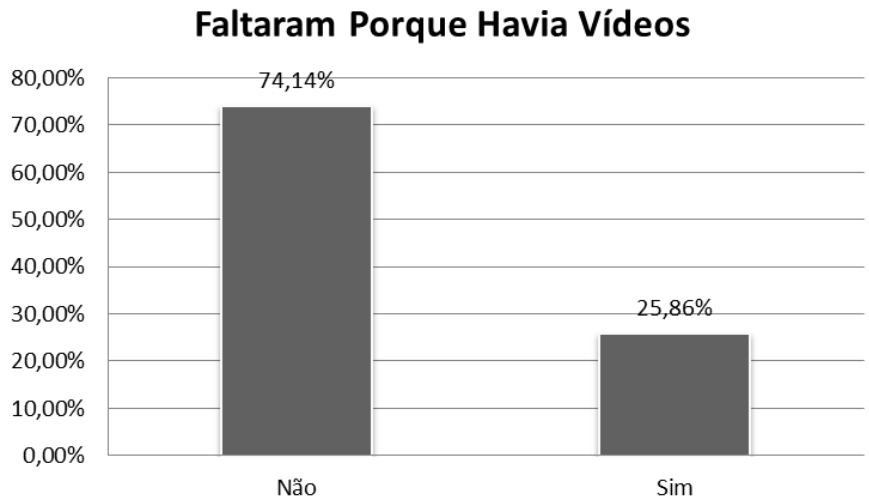


Figura 18 – Alunos que indicaram que faltaram aula por saber que o vídeo seria colocado online

Também foi perguntado aos alunos se eles assistiriam a uma aula remotamente em tempo real, mostrando o mesmo vídeo que era produzido nos *screencasts*. Surpreendentemente, 48% dos alunos responderam que assistiriam a uma aula remotamente, caso esta fosse transmitida via WEB. Esse resultado abre portas para alguns testes como, por exemplo, aulas de tutoria virtual em que a presença não é obrigatória e afins. Como não é o escopo deste trabalho, não foram realizados testes nesse sentido.

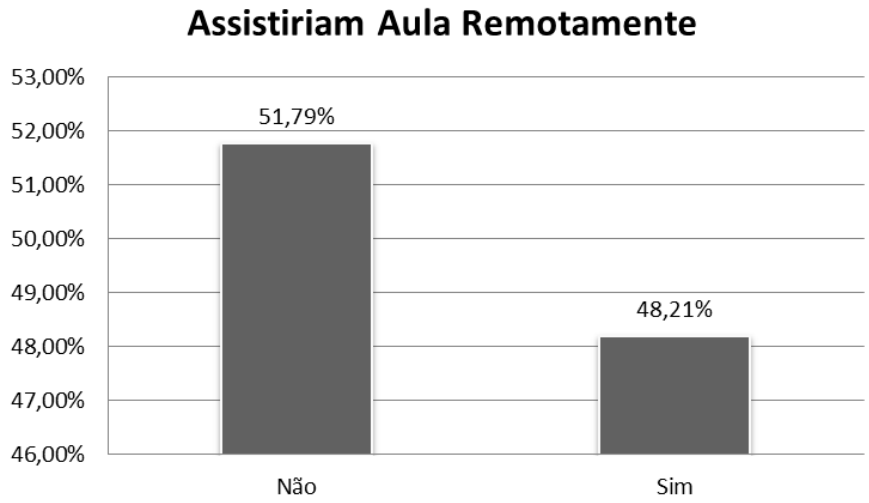
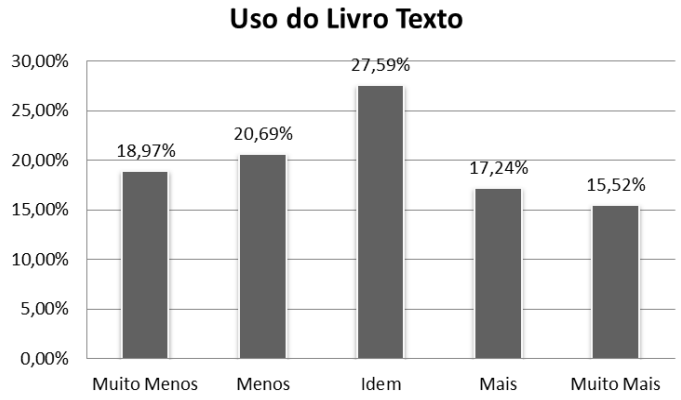
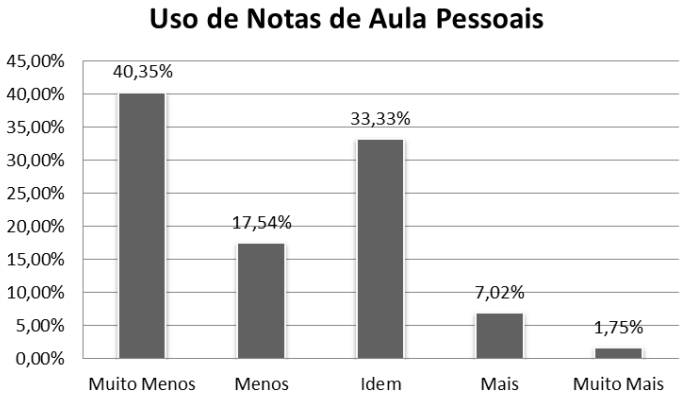


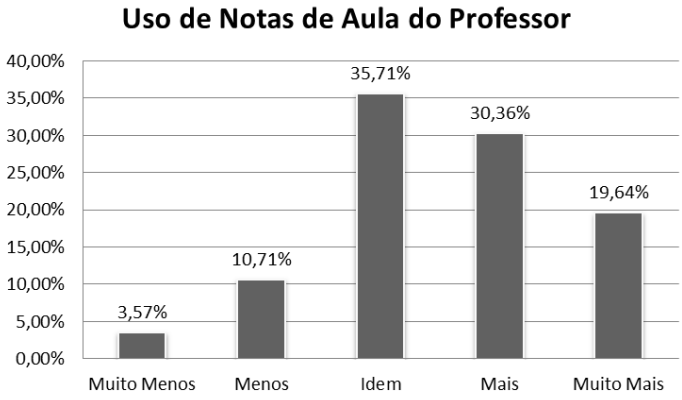
Figura 19 – Percentagem de alunos que assistiriam a aula via WEB

Com relação ao método de estudo dos alunos, achou-se interessante avaliar quais materais eles utilizaram para estudar durante o curso, uma vez que agora eles tiveram à sua disposição os *screencasts* e também os *slides* anotados do professor. Foi pedido aos alunos que eles comparassem, em relação com outras disciplinas que não utilizam o Tablet PC, o uso de livro texto, lista de exercícios, notas de aula pessoais, notas de aula do docente e de outros, como a Internet. A figura 20 mostra os resultados obtidos.

Podemos ver que, em relação ao livro texto e lista de exercícios, houve uma boa distribuição dos alunos, sendo que podemos concluir que as modificações introduzidas no curso não afetaram, pelo menos não de maneira sistemática, o uso desses recursos didáticos na rotina de estudo dos alunos. Essa tendência não se repete em relação ao uso de notas de aulas pessoais, com 58% dos alunos citando que fez menor uso de suas anotações pessoais e apenas uma pequena parcela (9%) respondendo que fez mais ou muito mais uso de anotações pessoais, em comparação a cursos que não utilizaram Tablet PC. O uso de notas de aula do professor passou a ser mais ou muito mais utilizado por 50% dos alunos, indicando que frente à facilidade de consulta, os estudantes consideram esse material uma boa fonte de conteúdo.


Uso do Livro Texto
30,00%
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
18,97%
20,69%
27,59%
17,24%
15,52%
Muito Menos
Menos
Idem
Mais
Muito Mais


Uso de Notas de Aula Pessoais
45,00%
40,00%
35,00%
30,00%
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
40,35%
17,54%
33,33%
7,02%
1,75%
Muito Menos
Menos
Idem
Mais
Muito Mais


Uso de Notas de Aula do Professor
40,00%
35,00%
30,00%
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
3,57%
10,71%
35,71%
30,36%
19,64%
Muito Menos
Menos
Idem
Mais
Muito Mais

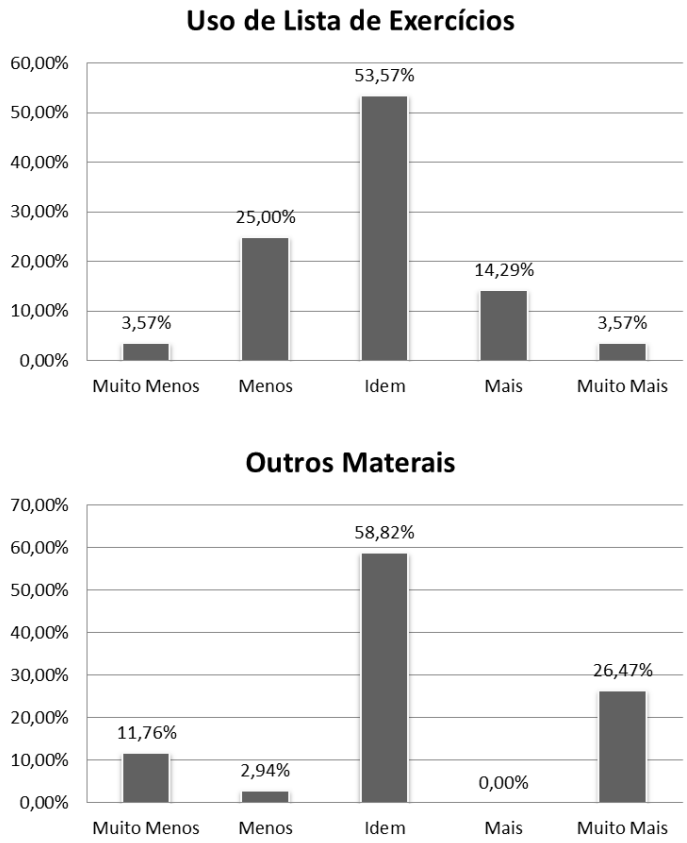


Figura 20 – Comparativo do uso de materais didáticos no curso

Além das medições explícitas, através dos questionários, os vídeos permitem a utilização de medidas implícitas, como o *log* de acesso do servidor WEB, para levantarmos algumas informações relevantes sobre o comportamento dos alunos. A figura 21 mostra um gráfico do tráfego de dados no servidor no mês de outubro. O eixo Y mostra a quantidade de dados transferidos (em MB), enquanto o eixo X mostra os dias do mês de Outubro.

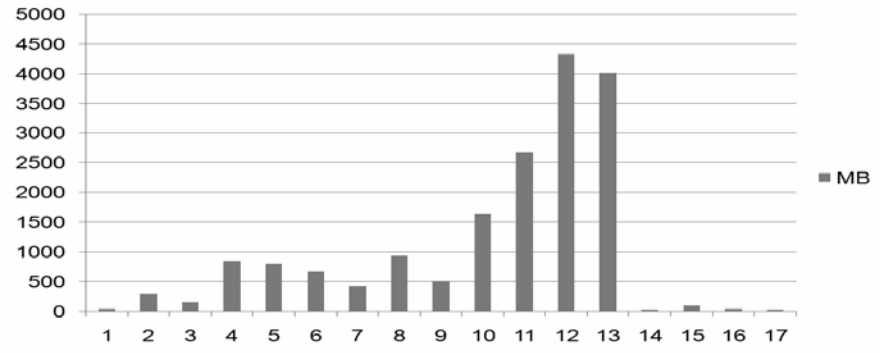


Figura 21 – Tráfego de dados no servidor, mês de Outubro

Foi aplicada uma prova no dia 13 de Outubro. Como podemos ver, há um aumento sistemático no tráfego de dados a partir do dia 9 de Outubro, mostrando que os alunos começam a estudar para a prova com cerca de 4 dias de antecedência. Esse padrão se repetiu para a segunda prova, com o tráfego do servidor aumentando drasticamente 4 dias antes da prova.

Outro dado interessante é a quantidade de acessos únicos às aulas. A figura 22 ilustra o número de acessos únicos a cada uma das aulas. O gráfico mostra no eixo Y o número de alunos e no eixo X a data da aula. Olhando a correspondência da data com o conteúdo, podemos levantar que as aulas mais acessadas foram as referentes a *Hazards* e Memória Virtual. Isso permite que o docente tenha uma noção de quais tópicos os alunos estão tendo mais dificuldade. Numa eventual dúvida sobre o que explicar numa aula de revisão, ou qual tópico gastar mais tempo, o professor tem uma indicação mais concreta de quais pontos não tenham ficado claros para os alunos.

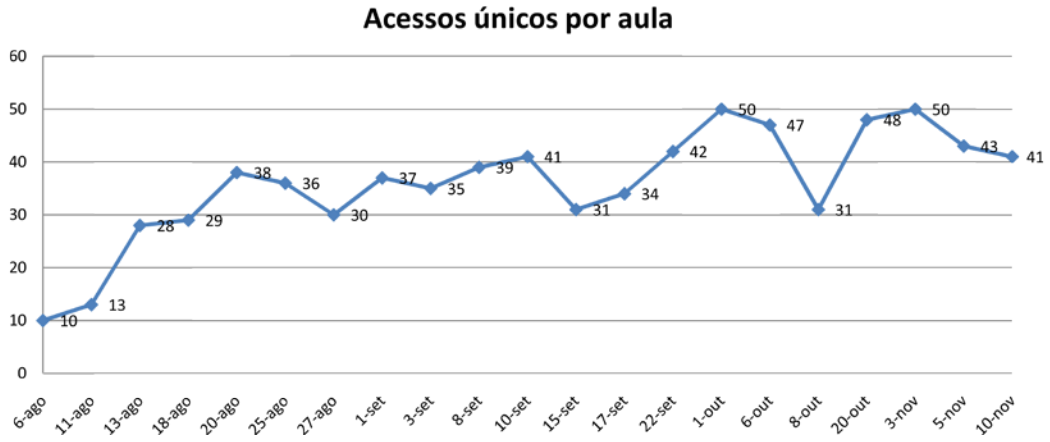


Figura 22 – Número de acessos únicos para cada aula

4.4. Resultados do uso do Tablet PC para apresentação da lousa e para gravação de *screencasts*: visão do docente

O modelo adotado de *screencast* permite que os alunos assistam, novamente, exatamente a mesma aula que assistiram em sala. Desta forma, eles têm maior controle sobre a navegação nos vídeos, movimentando para os momentos desejados das aulas e tirando dúvidas específicas no momento que desejarem. Um exemplo disto é o padrão de acesso aos vídeos, o qual possui altas taxas de transferência do servidor durante a noite e madrugada. Também sobre as taxas de transferências do servidor, é possível distinguir as aulas mais assistidas e, com isto, fornecer mais exercícios para sanar as dúvidas dos alunos.

Outro ponto importante a considerar é que o trabalho de gravação é extremamente simples, bastante um microfone de mesa e software para tal fim no Tablet PC.

5. Conclusões

A utilização de Tablet PCs no ensino de Arquitetura de Computadores resultou num melhor aproveitamento do curso por parte dos alunos. As mudanças introduzidas cumpriram seus objetivos principais, que eram aumentar a participação dos alunos, dar um retorno em tempo real para o professor sobre o entendimento dos estudantes, melhorar a comunicação entre alunos e professor e fornecer um material extra-classe que fosse útil aos alunos.

A primeira preocupação era se a introdução da nova metodologia não ia ser por demais complicada. Ao avaliarmos a dificuldade dos estudantes para se adaptar ao ambiente, encontramos uma pequena porcentagem de estudantes que tiveram algum tipo de dificuldade.

A maior participação dos alunos ficou evidenciada pelos resultados de maior participação individual e em grupo, mostrando que os alunos perceberam que suas ações tiveram um efeito direto no decorrer das aulas.

Houve também uma melhora na percepção que os alunos têm sobre o acesso ao docente para esclarecimento de dúvidas. Uma vez que o docente tinha acesso às soluções de exercícios dos alunos, ele podia avaliar erros comuns e discutí-los com a classe. O docente também teve a oportunidade de mostrar mais de uma solução para um problema, inclusive soluções erradas, trabalhando assim diversas maneiras diferentes de se chegar ao resultado correto.

A preferência pelos Tablet PCs, ao invés de papel e caneta, mostra a importância de tais dispositivos no processo de aprendizado, tornando-os um bom investimento para aprimorar a qualidade do ensino. Isso requer, no entanto, algumas modificações no curso, como a inclusão de exercícios voltados para essas atividades. Os exercícios, entretanto, devem ser mesclados: os mais curtos (e numerosos) para fixação do conteúdo, e os mais longos (e em menor quantidade) para a exploração de conceitos mais elaborados.

Outro ponto validado nos resultados é a necessidade de pelo menos 1 Tablet PC para cada dois alunos, no caso de uma aula ativa e colaborativa. Numa turma de 80 alunos, isso requereria pelo menos 40 Tablets, que podem ter um custo proibitivo para muitas instituições de ensino. Um número menor de Tablet PCs pode levar a um nível alto de distração, fazendo com que parte dos benefícios trazidos por uma aula ativa e colaborativa sejam perdidos.

Em relação à nova maneira de apresentação e uso dos *screencasts*, podemos concluir que são modificações extremamente bem sucedidas, ainda mais pelo fato de sua facilidade de implantação. Nesse caso, ao contrário das aulas ativas e colaborativas, basta apenas um Tablet PC e software adequado, sendo o curso ministrado praticamente sem alterações. As alterações necessárias se devem ao fato do instrutor ter o recurso de utilizar a tinta digital sobre as transparências, eliminando assim a necessidade de ir à lousa para mostrar deduções ou ter de dividi-las em várias transparências.

Os vídeos provocaram uma boa audiência e não tiveram o efeito colateral de espantar os alunos das aulas, pelo contrário. Em comparação com semestres anteriores, a taxa de reprovação por faltas caiu. Mesmo assim, em casos especiais, os alunos faltaram à aula

porque sabiam que nessa disciplina haveria um vídeo da aula como material extra. Isso permite que o aluno realize suas atividades extra-classe sem grandes prejuízos, podendo depois assistir ao vídeo da aula no momento mais conveniente.

Houve poucas reclamações referentes à qualidade e disponibilidade dos vídeos. A maioria das reclamações referia-se à falta de áudio das perguntas dos alunos e à qualidade de áudio. Os vídeos também fornecem ao professor maneiras indiretas de medir as práticas de estudo dos alunos, destacando quais pontos ficaram menos claros durante o semestre.

Como trabalho futuro, planeja-se uma forma de introduzir índices nos vídeos, a fim de facilitar a visualização de partes específicas dos mesmos. Outro ponto é a elaboração de experimentos que permitam uma avaliação quantitativa, verificando se há benefícios diretos do uso do Tablet PC no desempenho dos alunos. Há também a necessidade de consolidar quais exercícios são mais adequados para aplicação em aulas ativa e colaborativas.

6. Referências

[1] Anderson, R., Anderson, Ruth, Chung, O., Davis, K. M., Davis, Peter, Prince, C., Razmov, V., Simon, Beth (2006) "Classroom Presenter: A Classroom Interaction System for Active and Collaborative Learning", The Impact of Tablet PCs and Pen-based Technology on Education: Vignettes, Evaluations and Future Directions, Purdue University Press, p.21-30

[2] Backon, Joel (2006) "Student Minds and Pen Technologies: A Wonderful Pedagogical Marriage", The Impact of Tablet PCs and Pen-based Technology on Education: Vignettes, Evaluations and Future Directions, Purdue University Press, p.1-11

[3] Bransford, J., Brown, A., Cooking, R. (2000) "How People Learn: Brain, Mind, Experience and School", National Academy Press

[4] Carryer, J. Edward (2006) "The Tablet PC as a Platform for Screencasting Lectures", The Impact of Tablet PCs and Pen-based Technology on Education: Vignettes, Evaluations and Future Directions, Purdue University Press, p.41-48

[5] Crooks, Clayton E. (2004) – Developing Tablet PC Applications – Charles River Media

[6] Hennessy, John L., Patterson, David A., Goldberg, David (2002) "Computer Architecture: A Quantitative Approach, Third Edition", Morgan Kaufmann, 3rd edition

[7] Jarrett, R., Su, Philip (2002) - Building Tablet PC Applications - Microsoft Press

[8] Koile, Kimberle, Singer, David (2006) "Development of a Tablet PC Based System to Increase Instructor-Student Classroom Interactions and Student Learning", The Impact of Tablet PCs and Pen-based Technology on Education: Vignettes, Evaluations and Future Directions, Purdue University Press, p.115-122

[9] McConnell, J. (1996) Active Learning And Its Use in Computer Science. In Integrating Technology into CSE

[10] Price, Edward, Simon, Beth (2007) "A Survey to Assess the Impact of Tablet PC Based Active Learning: Preliminary Report and Lessons Learned", The Impact of Tablet PCs and Pen-based Technology on Education: Beyond the Tipping Point, p.97-105

[11] Prince, M. (2004) Does Active Learning Work? A review of the research. In Journal of Engineering Education, 93(3), 223-232

[12] Simon, Beth; Anderson, Ruth; SU, Jonathan. (2004) "Preliminary Experiences with a Tablet PC Based System to Support Active Learning in Computer Science Course." In Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education Session: Technology in CS education table of contents, p213-217, United Kingdom

[13] Subhlok, J., Johnson, O., Subramaniam, V., Vilalta, R., Yun, C. (2007) "Tablet PC video based hybrid coursework in computer science: report from a pilot project", ACM SIGCSE Bulletin Issue 1, p.74-78

Tablet PC Review - http://www.tabletpcreview.com

Capítulo

2

Ensino de Arquitetura de Computadores: Uma Abordagem Utilizando a Metodologia de Aprendizagem Baseada em Problemas

Wagner L. A. de Oliveira, Anfranserai M. Dias, Antonio L. Apolinário Jr., Angelo A. Duarte, Tiago de Oliveira

E-mails: {*wagner*|*anfranserai*|*apolinario*|*angeloduarte*|*tiagoooli*}@ecomp.uefs.br
Endereço: Departamento de Tecnologia
Universidade Estadual de Feira de Santana
Avenida Transnordestina, s/n
Novo Horizonte - Feira de Santana - BA
CEP - 440360-900
tel: (75)3224-8056

Resumo

Esse capítulo tem por objetivo apresentar a experiência do grupo de professores do curso de Engenharia de Computação da Universidade Estadual de Feira de Santana no uso da metodologia Problem-Based Learning - PBL *no ensino das disciplinas relacionadas a área de Arquitetura de Computadores. Após uma breve contextualização da metodologia PBL, discutiremos a forma como esta vem sendo aplicada dentro do curso de Engenharia de Computação. A partir de quatro estudos de casos, analisaremos os resultados do uso de PBL no ensino das disciplinas vinculadas à área de Arquitetura de Computadores, indicando seus principais ganhos diante das metodologias de ensino tradicionais.*

2.1. Introdução

A adoção de uma abordagem de ensino utilizando a metodologia de aprendizagem baseada em problemas, comumente chamada de PBL, no curso de Engenharia da Computação da UEFS foi motivada, na época, pelas discussões promovidas por ministérios e associações ligadas à educação e relacionadas ao perfil profissional de um engenheiro e a sua inserção na sociedade (Bittencourt & Figueiredo, 2003) (Bittencourt *et al.*, 2002). Desta

forma, para podermos compreender os verdadeiros motivos que levaram os fundadores e colaboradores do curso a adotarem o método PBL, faremos, na seqüência, um breve relato do panorama existente na época.

Desde o final do século XX vem ocorrendo vários debates relacionados às competências, habilidades e atitudes que um engenheiro deve possuir para poder exercer adequadamente sua prática profissional no mundo atual (de Ensino de Engenharia, 1991). Estes debates foram fortalecidos pela iniciativa do Ministério da Educação (MEC) de reformular as diretrizes curriculares dos cursos de graduação (dos Santos *et al.*, 2007). No caso específico da engenharia, o debate envolvendo a Associação Brasileira de Ensino de Engenharia (ABENGE), o sistema CREA/CONFEA de regulamentação e fiscalização das atividades dos profissionais de engenharia, as instituições de ensino superior e a Comissão de Especialistas de Ensino de Engenharia do MEC (CEEEng/MEC) levou a aprovação da resolução 11/2002 pelo Conselho Nacional de Educação (CNE) em 2002 (CNE, 2002). Esta resolução instituiu diretrizes curriculares nacionais envolvendo o curso de graduação em engenharia e uma série de competências gerais foi levantada nos incisos do artigo quarto desta resolução, dentre as quais podemos destacar (CNE, 2002) (dos Santos *et al.*, 2007):

- "V – identificar, formular e resolver problemas de engenharia";
- "VIII – comunicar-se eficientemente nas formas escrita, oral e gráfica";
- "IX – atuar em equipes multidisciplinares" e
- "XIII – assumir a postura de permanente busca de atualização profissional".

No entanto, nos cursos mais convencionais, as competências mencionadas acima não costumam ocorrer de forma satisfatória, visto que, segundo a própria ABENGE, os cursos tradicionais são baseados em conhecimento, com enfoque no conteúdo e centrado no professor (dos Santos *et al.*, 2007).

Diante deste cenário, a decisão dos fundadores do curso de Engenharia da Computação da UEFS de adotar o método PBL, ocorreu durante a elaboração do projeto curricular. Neste projeto curricular, o método PBL foi considerado como uma das possíveis metodologias a serem utilizadas na tentativa de reforçar a interação entre teoria e prática, a qual poderia ser alcançada por meio de um ciclo denominado situação-fundamentação-realização (Bittencourt *et al.*, 2002) (dos Santos *et al.*, 2007).

Na fase inicial deste ciclo (etapa de situação), apresenta-se ao aluno um problema ou projeto, normalmente, do mundo real, procurando mantê-lo em contato com fenômenos e objetos que o motive a adquirir novos conhecimentos técnicos para a resolução do problema proposto. Na segunda etapa, ocorre a fundamentação em termos da aquisição de conceitos teóricos necessários à resolução dos problemas previamente identificados na etapa de situação. Diferentemente do ciclo convencional de ensino, onde os conceitos teóricos são estudados antes da apresentação de qualquer problema, o aluno, nesta nova metodologia de aprendizado, deve realizar todo o levantamento bibliográfico necessário para a resolução do problema, iniciando uma reflexão crítica sobre o problema aplicado.

O problema deve ser capaz de despertar no aluno a motivação para que este tenha interesse suficiente para adquirir a base teórica que lhe falta para compreender e solucionar os problemas propostos. Por fim, na fase final deste ciclo (etapa de realização), o aluno deve utilizar os conceitos teóricos estudados para a produção de um produto final, aproximando a teoria aprendida com a prática, e permitindo ao aluno a compreensão da realidade apresentada. A realização de soluções para os problemas estudados anteriormente sedimenta definitivamente a compreensão do assunto.

Ao analisarem diversas metodologias de aprendizagem ativa, os fundadores do curso verificaram que o método PBL seria uma excelente candidata para aplicação do ciclo de situação-fundamentação-realização no contexto de um componente curricular de duração semestral (dos Santos *et al.*, 2007). Outras metodologias que implementam este ciclo tais como aprendizagem por projetos e estudo independente (McKeachie, 1999) são também utilizadas em outros componentes curriculares do curso, como projetos anuais e trabalho de conclusão de curso.

Após essa breve descrição dos motivos que levaram os fundadores do curso de Engenharia da Computação da UEFS de adotar a metodologia de aprendizado PBL, apresentamos, na seqüência deste capítulo, a experiência do grupo de professores deste curso no uso do método PBL no ensino das disciplinas relacionadas à área de Arquitetura de Computadores.

Inicialmente, na seção 2.2, apresentamos a fundamentação teórica da metodologia PBL, descrevendo brevemente, o histórico da aplicação do método no mundo, os seus objetivos, conceitos fundamentais da metodologia, formas de avaliação e a correlação entre o método PBL e a engenharia. Após a fundamentação teórica, na seção 2.3, abordamos os conceitos do método PBL sobre a seleção de tópicos, elaboração de problemas/projetos e formas de avaliação aplicados no curso de Engenharia da Computação da UEFS, mais especificamente, no Estudo Integrado de Sistemas Digitais, o qual engloba as disciplinas de Arquitetura de Computadores e Arquitetura de Computadores Avançada. Na seção 2.4, apresentamos quatro estudos de casos de problemas/projetos que já foram previamente aplicados no Estudo Integrado de Sistemas Digitais, a saber: Projeto de um processador, Sistema de Memória, Mecanismos de Entrada/Saída e Seminários. Em cada um dos casos, descrevemos os conceitos que foram trabalhados durante a aplicação do problema/projeto, as dificuldades no seu desenvolvimento e os resultados obtidos em relação ao desempenho dos grupos tutoriais de alunos. Por fim, na seção 2.5, realizamos as considerações finais sobre a aplicação da metodologia PBL no curso de Engenharia da Computação da UEFS.

2.2. Fundamentação Teórica

Essa seção descreverá as bases teóricas de PBL e citará os principais autores e textos usados para explicar essa metodologia. A seção se dividirá nos tópicos: Histórico, Objetivos, Metodologia e Formas de Avaliação. A seção será uma fonte de referências, que permitirá ao leitor desenvolver habilidades na implementação da metodologia em sua prática. Dessa forma, pretende-se familiarizar o leitor com os fundamentos de PBL, de maneira que o mesmo possa avaliar melhor como tal metodologia tem sido aplicada no curso de Engenharia de Computação da UEFS e, por conseguinte, possa ser incentivado a incluir

essa prática também em sua disciplina.

2.2.1. Histórico

As primeiras referências ao uso de PBL remontam ao final da década de 1960, quando o método foi aplicado na Faculty of Medicine at the McMaster University (Canada) tomando como base as experiências de ensino de direito da Escola de Direito da Universidade de Harvard (nos anos 1920) e no método de ensino de medicina usado na Case Western Reserve University (nos anos 1950), ambas nos EUA (Costa *et al.*, 2007) (Montero & Gonzalez, 2009) (Mantri *et al.*, 2008) (Chang & Lee, 2006).

Ainda que muito já tenha sido escrito sobre PBL, não se pode definir uma fundamentação teórica explicita para a sua origem pelo simples fato de que essa fundamentação não foi estabelecida pelos seus idealizadores. Entretanto, o conceito de aprendizagem motivada pela necessidade de solução de problemas pode ser atribuído a Dewey, o qual definiu o princípio da aprendizagem autônoma (Schmidt, 1993) (Penaforte, 2001) (Woods, 1995). Uma análise conceitual dos efeitos do PBL e suas relações com as teorias de aprendizagem de Dewey, Piaget e outros pode ser obtida no trabalho de Douchy et. al (Dochy *et al.*, 2003).

Uma extensa síntese dos estudos sobre o método PBL pode ser encontrada no trabalho de Albanese e Mitchell (Alabanese & Mitchell, 1993), a qual baseou-se num estudo analítico/comparativo dos resultados de 41 pesquisas sobre resultados do método PBL em escolas de medicina da Austrália, Canadá, Estados Unidos e Holanda. Os autores analisaram questões sobre custos de uso do método PBL, impactos na carga horária e na satisfação dos docentes, volume de conhecimento aos quais os alunos eram expostos, desenvolvimento da capacidade cognitiva dos alunos e a dependência criada pelo modelo de aprendizagem em grupo sobre os profissionais formados via PBL.

Já Vernon e Blake (Vernon & Blake, 1993) realizaram uma meta-análise estatística e buscaram uma comparação quantitativa entre o PBL e os métodos convencionais. Um trabalho similar foi realizado por Dochy et al. (Dochy *et al.*, 2003). Neste último, foram abordadas questões sobre os efeitos do PBL sobre conhecimentos e habilidades dos estudantes.

Ribeiro (Ribeiro, 2008b), defende que os resultados das pesquisas de Albanese e Mitchell, Vernon e Blake e Dochy et. al. Convergem independentemente das diferentes metodologias utilizadas. Há, segundo Ribeiro, uma posição claramente favorável dos alunos quanto à metodologia nas três análises. Existem ainda concordância nas pesquisas sobre a preferência dos alunos quando podem escolher entre PBL e métodos convencionais. As pesquisas apontam para um desempenho superior dos alunos de PBL em testes de seus conhecimentos práticos quando comparados a alunos de métodos convencionais, o que é confirmado por notas superiores dos alunos de PBL em avaliações de seus superiores em estágios.

Ribeiro salienta ainda que as pesquisas de convergem em relação à opinião dos docentes. Segundos essas pesquisas, os professores reconhecem o aumento do tempo dedicado à docência, mas consideram o método PBL mais agradável que o método convencional. Entretanto, Ribeiro também aponta divergências entre as pesquisas com relação

ao desempenho dos alunos, medido por meio de testes objetivos, ser igual o inferior ao dos alunos de métodos convencionais.

Solomon (Solomon, 2003) apresenta um estudo indicando que, em um contexto educacional multidisciplinar, PBL se destaca como uma das estratégias de ensino mais interessantes já que é induz o estudante a encontrar inter-relações entre conceitos presentes em diversos conjuntos de conhecimento.

2.2.2. Objetivos

O objetivo do PBL é desenvolver de forma integrada um conjunto de conhecimentos, habilidades e atitudes através da solução de problemas reais usando tanto aprendizagem autônoma (indivíduo) quanto colaborativa (equipes e grupos).

Dessa forma, o PBL procura atingir simultaneamente os objetivos de aprendizagem ativa, aprendizagem integrada, aprendizagem cumulativa e aprendizagem para a compreensão (Hadgraft, 1998) (Hadgraft & Holecek, 1995). A aprendizagem ativa se dá pela elaboração de respostas às perguntas do problema. A aprendizagem integrada é conseguida pela elaboração de problemas que exijam o conhecimento de diversas subáreas para a elaboração de suas respostas. A aprendizagem cumulativa se consegue escalando gradativamente a complexidade dos problemas até que se chegue a problemas tão complexos quanto aqueles enfrentados por um profissional que inicia sua carreira. Finalmente, a aprendizagem para a compreensão é conseguida dando-se tempo para reflexão sobre as respostas, retroalimentando os alunos sobre seu desenvolvimento e criando oportunidades para a aplicação prática dos conhecimentos (Komatsu *et al.*, 1998).

Não é suficiente apenas aprender questões técnicas para passar em testes e avaliações, mas as técnicas necessárias para enfrentar situações reais. O método PBL procura enfatizar as conexões entre diferentes aspectos do mundo profissional em que o estudante atuará, estimulando uma visão abrangente dos problemas para ilustrar as restrições quanto aos aspectos técnicos, tecnológicos, humanos e sociais da solução de problemas do mundo real.

2.2.3. Metodologia

A Aprendizagem Baseada em Problema é um método de aprendizagem no qual o problema comanda o processo de aprendizagem. Em outras palavras, o problema precede o conhecimento do estudante.

O problema é posto de forma que o estudante descubra que ele precisa aprender algum conhecimento antes que eles possam solucionar o problema. Aprender dentro de um contexto onde há um problema a resolver tende a motivar os estudantes e também promove o armazenamento do conhecimento em padrões de memória que facilitam sua recuperação posterior para a solução de novos problemas (Groh & Duch, 2001).

Ambientes PBL geralmente incluem projetos de pesquisa, casos clínicos, bem como problemas de projetos em engenharia que são mais que uma simples síntese de conhecimentos previamente aprendidos. Essa abordagem demanda mudanças nos métodos de ensino, particularmente nos professores que terão que deixar de ser o centro das atenções e fonte de conhecimento e passar a ser treinadores e facilitadores do processo de

aquisição do conhecimento. A aprendizagem torna-se centrada do estudante em lugar de centrada no professor (Felder & Brent, 2003) (Komatsu *et al.*, 1998).

A estratégia do PBL tenta engajar estudantes em autenticas tarefas do mundo real para aumentar a eficiência do aprendizado. Os estudantes são tipicamente organizados em grupos e enfrentam projetos com o instrutor fazendo o papel de treinador e facilitador. No caso das engenharias, cada time projeta e constrói um sistema completo. Na medicina os estudantes diagnosticam, definem o tratamento e, eventualmente, acompanham o desenvolvimento do paciente. Em administração definem estratégias, preparam apresentações e relatórios sobre abordagens para problemas organizacionais. Seja qual for a área do conhecimento, o ambiente de aprendizagem deve simular situações profissionais nas quais os estudantes tenham que trabalhar com diferentes tipos de conhecimento.

Os projetos devem adotar uma ampla gama de habilidades, não somente aquelas relacionadas ao conhecimento ou questões técnicas, mas também habilidades práticas, tais como:

- Deparar-se com informações imprecisas ou incompletas: Não raro, problemas do mundo real contêm uma quantidade insuficiente ou imperfeita de informações. Em medicina um paciente pode confundir-se ou ser impreciso ao dar uma informação ao seu médico. Em engenharia muitas vezes deve-se lidar com incertezas sobre as informações do ambiente onde um produto irá ser utilizado.

- Comprometimento e auto-regulação: Os estudantes envolvem-se mais no processo de aprendizagem quando eles devem definir seus próprios objetivos dentro das restrições impostas pelas regras gerais definidas pelos instrutores dos cursos. Uma das restrições mais comuns e sempre presentes no mundo real refere-se ao tempo em que um projeto deve ser concluído.

- Trabalho cooperativo: Os estudantes devem sempre se organizar para dividir a tarefas entre si e para posteriormente consolidar os trabalhos individuais em um único trabalho.

- Relações interdisciplinares: Problemas complexos envolvem conhecimentos ministrados em diversas disciplinas. Na metodologia de ensino tradicional centrada em aulas, na qual, via de regra, os professores focalizam um assunto específico e abstraem o efeito do contexto. Diferentemente, no PBL as questões interdisciplinares são inevitáveis já que os estudantes acabam lidando com problemas comuns do contexto profissional em lugar de lidar com abstrações.

Segundo Thomas et al. (Thomas *et al.*, 1999), existem cinco critérios que um projeto deve atender para ser considerado um problema adequado ao uso em PBL:

- Deve estar centralizados no curriculum do curso;

- Deve focar em questões que induzam os estudantes a encontrar os conceitos centrais;

- Suas atividades devem envolver a construção dos conhecimentos dos estudantes;
- Deve ser conduzidos pelos estudantes;
- Deve ser realista e não abstrato.

2.2.4. Formas de Avaliação

Avaliar é sempre uma atividade crítica em qualquer método de aprendizagem. Encontrar o equilíbrio entre a exigência que deve ser feita no processo avaliativo e a correta mensuração da evolução do estudante, ou em outras palavras, do seu aprendizado, é uma atividade que exige uma grande dose de dedicação dos docentes. No PBL essa dificuldade não é menor que no método tradicional, mas permite ao docente lidar com mais possibilidades de avaliação do que apenas as provas, relatórios e trabalhos típicos dos métodos convencionais.

Bruner (Bruner, 1973) e Ramos (Ramos, 1999) indicam que o processo de avaliação deve atender às necessidades de alunos, professores e especialistas das disciplinas levando em consideração que o processo de avaliação em um ambiente de trabalho cooperativo deva ser realizado pelos alunos, por seus pares e pelo tutor.

Tipicamente, o problema PBL demanda o desenvolvimento de um produto na forma de um relatório, apresentação, artefato, programa de computador ou qualquer outro elemento que esteja em concordância com a disciplina que está sendo estudada. No método PBL a avaliação do estudante é um processo contínuo que se inicia no primeiro dia. O professor deve atentar para aspectos de conhecimento técnico do assunto, de habilidades na solução de problemas e trabalho em grupo e nas atitudes durantes as atividades realizadas (Amador *et al.*, 2007) (Schwartz, 2001).

É consenso na literatura que uma parte do rendimento dos alunos seja avaliada através de auto-reflexões e auto-avaliações. Os modelos de auto-avaliação a serem seguidos variam ligeiramente entre si, e de forma geral procuram fazer com que o aluno avalie sua contribuição com idéias e fatos para a solução do problema, a forma como colaborou para que o grupo chegasse à solução do problema e os recursos usados para a obtenção de informação. Como a auto-avaliação geralmente põe os alunos em uma posição difícil, principalmente para aqueles alunos que ainda são noviços no estudo com PBL, é recomendável que no início o professor auxilie com um modelo ou definindo de forma direta os critérios de auto-avaliação. Eventualmente, com o amadurecimento dos alunos estes mesmos podem passar a definir critérios próprios para si e para o grupo.

O professor também tem oportunidade de se auto-avaliar no processo de guiar os estudantes. Delisle (Delisle, 1997) sugere um conjunto de critérios de avaliação que o professor pode usar para analisar seu próprio desempenho na tutoria dos grupos. Esse conjunto de critérios procura avaliar se o professor cria um ambiente propício para os desenvolvimento dos trabalhos, se os alunos foram encorajados a refletir sobre o conhecimento necessário para a solução do problema, além de critérios quanto a infra estrutura e os recursos usados nos trabalhos.

A eficiência do problema deve ser questionada ao final de cada problema usando

uma avaliação que contemple, ainda conforme Deslile (Delisle, 1997), um conjunto básico de questões. Essas questões procuram avaliar se o problema:

- atingiu os objetivos curriculares;
- desenvolveu habilidades de pensamento e raciocínio nos estudantes;
- representou bem problemas do mundo externo;
- se relaciona com interesses dos estudantes, ou desperta interesse nos estudantes;
- tem um nível de complexidade adequado;
- era possível de ser solucionado com os recursos disponibilizados aos estudantes;

Esse é um conjunto de questões básico e pode ser acrescido de outros que sirvam para adequar a avaliação à realidade da instituição. As respostas a estas questões servirão como base para eventuais ajustes no problema para os próximos grupos.

2.2.5. PBL e Engenharias

O método PBL possui uma sintonia natural com a engenharia já que em sua vida prática um engenheiro freqüentemente se depara com situações nas quais os problemas enfrentados apresentam dificuldades que precisam ser contornadas com novas tecnologias ou com a combinação inovadora de tecnologias existentes. Tal qual na medicina, a engenharia também se vale da seqüencia Hipótese $\rightarrow$ Diagnóstico $\rightarrow$ Tratamento quando o engenheiro realiza trabalhos de manutenção em equipamentos ou estruturas e nesse sentido, seu trabalho se assemelha com o de um médico ao tratar um paciente.

Não é de espantar, portanto, que o método PBL tenha rapidamente se espalhado pelo ensino das engenharias. Um trabalho relatando o uso de PBL no curso Engenharia Elétrica e Engenharia de Comunicações da *Helsinki University of Technology* (Finlândia) é descrito por Costa, Honkala e Lehtovuori (Costa *et al.*, 2007). Estes autores relatam o uso de PBL desde 1999 na disciplina de Laboratório de Teoria de Circuitos e declaram que o método PBL tem uma vantagem pequena em relação ao método tradicional quando são comparadas as notas médias dos alunos. Eles afirmam que a maior importância do PBL reside criação de uma boa atmosfera de aprendizagem pelo reforço da autoconfiança dos estudantes que passam a ter maior habilidade crítica e de questionamento.

Outro trabalho, realizado por Chang e Lee (Chang & Lee, 2006), descreve a incorporação de PBL no curso de Ciência da Computação na *National Taiwan Normal University* (Taiwan), também relata a melhoria no ambiente de sala de aula. O trabalho foi realizado com 605 estudantes em 13 turmas e ao final os autores afirmam que houve uma mudança positiva na atitude de aprendizagem dos estudantes que passaram a ser agentes ativos do seu aprendizado. Segundo os autores o PBL melhorou os conceitos de ciência de computação as habilidades de uso de *softwares* dos estudantes.

Uma experiência do uso de PBL no *Chitkara Institute of Engineering and Technology* (Norte da Índia) é relatada por Mantri, Dutt, Gupta e Chitkara (Mantri *et al.*, 2008).

Os autores compararam os resultados obtidos por dois grupos de alunos da disciplina Eletrônica Analógica. Um grupo, chamado de grupo de tratamento (TG) aprendeu através do método PBL e um grupo de controle (CG) aprendeu pelo método tradicional. Os resultados apontaram um aumento significativo no desempenho dos alunos de PBL com relação às notas obtidas nas provas e exames de habilidades no final do semestre. Os autores argumentam que essa melhora foi causada por uma maior motivação dos alunos de PBL associada a um maior tempo de estudo que esses dedicaram no curso.

O ensino da disciplina Sistemas Operacionais usando PBL *Holon Institute of Technology* (Israel) é relatado por Pelleh et. al. (Pelleh *et al.*, 2008). Os autores apontam uma atitude positiva por parte dos estudantes e a melhoria em suas habilidades de colaboração e comunicação como resultados diretos do método PBL.

Um exemplo mais recente é o trabalho realizado por Montero e Gonzalez (Montero & Gonzalez, 2009) com a aplicação de PBL no estudo de Transferência de Calor para alunos do primeiro ano do curso de Graduação em Engenharia Eletrônica na *Escuela Politécnica Superior de la Universidad de Burgos* (Espanha). Os autores concluem que um dos fatores chaves para o sucesso do método no curso foi o grande interesse que os problemas despertaram nos alunos por conta de sua contextualização dentro da engenharia elétrica. Eles comentam que ainda que o método PBL cobrisse menos assuntos do que seriam possíveis com um método tradicional, os alunos obtiveram um aprendizado mais consistente e profundo do que foi apresentado, o que gerou resultados acadêmicos superiores aos conseguidos em anos anteriores. Eles comentam por fim alunos de primeiro ano precisam ser mais assessorados pelo tutor já que não possuem maturidade suficiente para abordar problemas abertos.

Diversas experiências positivas vêm sendo relatadas na literatura, indicando que o PBL é um método que aumenta a eficiência do processo de aprendizagem em engenharias. Essas experiências também apontam que o maior esforço e tempo despendido por professores e alunos se compensam pelo aprofundamento no aprendizado dos conceitos. Além do mais, em todos os casos são relatadas melhoras na atmosfera em sala de aula o que certamente contribui para um maior desempenho dos alunos.

2.3. PBL e Arquitetura de Computadores

Desde a primeira definição e aplicação do método PBL na escola de medicina da Universidade McMaster, este método vem sendo adaptado e utilizado em muitos contextos diferentes no ensino de diversas áreas do conhecimento (Santos *et al.*, 2008b) (Striegel & Rover, 2002). Essa adaptação para outros tipos de contextos de aprendizagem e de ensino resultou em formatos e abordagens alternativas ao modelo McMaster original (Martinez-Mones *et al.*, 2005) (Northern III, 2007). No modelo original apresentado por McMaster, um conjunto de problemas costuma representar o pilar principal dos elementos curriculares, sendo que os conhecimentos necessários para a sua solução são, em grande parte, buscados pelos próprios alunos, que trabalham em grupos de oito a doze indivíduos, conduzidos pelo tutor (Ribeiro, 2008a).

O modelo PBL original sofreu algumas modificações na sua aplicação em relação ao método PBL aplicado no Estudo Integrado de Sistemas Digitais da UEFS, pelo simples fato das soluções buscadas no ensino dessas áreas não se reduzirem à obtenção de um

diagnóstico e à escolha de um entre vários tratamentos. Na engenharia de computação, o processo de resolução do problema é mais complexo, frequentemente resultando em várias soluções possíveis, além de ser necessário a realização de um produto "real" ou concreto. Todo esse processo requer mais tempo e implica conhecimentos conceituais e procedimentais mais difíceis de serem desenvolvidos de forma autônoma em um tempo compatível com o período de formação dos alunos.

No estudo integrado de Sistemas Digitais da UEFS, a qual engloba os conteúdos de Arquitetura de Computadores, emprega-se o modelo PBL conhecido na literatura como modelo PBL parcial (Ribeiro, 2008a), como mostrado na figura 2.1. Neste modelo, o currículo tem vários componentes convencionais e um componente curricular no qual problemas/projetos são trabalhados por grupos de alunos sob supervisão de tutores. Neste caso, um conjunto de problemas é utilizado para introduzir, estruturar e aprofundar os conteúdos deste componente. Os conteúdos dos demais componentes (A, B, C e D) são trabalhados separadamente, empregando-se metodologias convencionais e desvinculados dos problemas apresentados no componente PBL. A principal limitação deste modelo está na probabilidade de que os vários componentes possam competir em relação à atenção e aos esforços dos alunos, especialmente em razão de seus diferentes ritmos de ensino e sistemáticas de avaliação de desempenho.

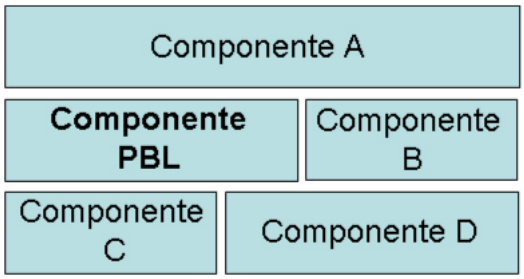


Figura 2.1. Modelo PBL parcial.

No estudo integrado de Sistemas Digitais da UEFS, o modelo parcial foi adaptado como mostrado na figura 2.2. Neste caso, existem dois componentes curriculares convencionais, a saber, Arquitetura de Computadores e Arquitetura de Computadores Avançada. Além desses dois componentes, o estudo integrado é composto de um componente PBL. Cada componente convencional dos componentes curriculares (i.e., módulos, matérias e laboratórios) dão suporte ao aluno para a resolução dos problemas especificados no componente PBL, cabendo aos docentes responsáveis por estes componentes convencionais a escolha da melhor metodologia para ensinar seus conteúdos (i.e., por meio de aulas expositivas, seminários, visitas externas, etc.). Uma vantagem deste modelo está na possibilidade dos recursos humanos e materiais serem previamente alocados a partir da demanda de conhecimentos prevista para os problemas.

Com relação à carga horária do estudo integrado, tem-se: quatro horas semanais para o componente curricular convencional "Arquitetura de Computadores", duas horas semanais para o componente "Arquitetura de Computadores Avançada" e duas horas semanais para o componente PBL. No componente PBL, as quatro horas semanais são divididas em duas sessões tutoriais, cada uma contendo duas horas semanais.

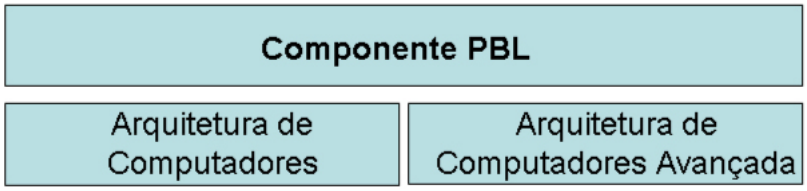


Figura 2.2. Modelo PBL parcial adotado no estudo integrado de Sistemas Digitais da UEFS.

2.3.1. Seleção de Tópicos

Nos dois componentes curriculares convencionais são ministradas aulas teóricas sobre os assuntos de sistemas digitais, mais especificamente, sobre arquitetura de computadores. Podemos citar os seguintes tópicos abordados no componente curricular "Arquitetura de Computadores":

1. Classificação de Computadores
2. Conjunto de Instruções
3. Modos de Endereçamento
4. Linguagem de Descrição de *Hardware* - Verilog
5. Unidade de Controle *Hardwired*
6. Unidade de Controle Microprogramada
7. Máquinas de Estados Finitos
8. Memória *Cache*
9. Dispositivos de Entrada/Saída

Normalmente, a ordem numérica acima equivale à ordem cronológica dos assuntos ministrados em sala de aula. Por sua vez, para o componente curricular "Arquitetura de Computadores Avançada", temos:

1. Classificação de Memórias
2. Sistema de Memória
3. Tipos de Memória
4. Dispositivos Lógicos Programáveis
5. Arquitetura *Pipeline*
6. *Pipeline*: Conflitos Estruturais e Dependências de Dados e de Controle
7. Conceitos Intermediários sobre *Pipeline*

8. Arquitetura Superescalar
9. Sistema de Interconexão
10. Sistema Multiprocessado
11. Memória Virtual

No componente curricular PBL, os tópicos necessários para a resolução dos problemas são os mais variados, dependendo, obviamente, do problema especificado e do grau de cobertura dos assuntos que se deve cobrir. Tem sido costume a aplicação de três ou quatro problemas durante o componente PBL, o qual tem duração de um semestre, com carga horária total de 60 horas. Os problemas iniciais aplicados aos alunos do componente PBL costumam estar relacionados aos itens 2, 3 e 7 do componente "Arquitetura de Computadores". Problemas típicos referem-se à confecção de programas em linguagem *Assembly*, utilizando uma biblioteca gráfica, como por exemplo, OpenGL, para o desenvolvimento de jogos. Ainda nesta linha de projetos, o tópico 9 também pode ser coberto com a implementação em FPGA de jogos utilizando uma linguagem de descrição de *hardware*, como Verilog ou VHDL, realizando a interface com dispositivos de entrada e saída, como teclados e monitores de vídeo. Após esses primeiros projetos, também costumam ser propostos projetos relacionados ao desenvolvimento de processadores RISC seqüenciais ou processadores que utilizem uma estrutura um pouco mais sofisticada, como pipeline. Desta forma, os tópicos 4, 5 e 6 do componente "Arquitetura de Computadores" e os tópicos 4, 5 e 6 do componente "Arquitetura de Computadores Avançada" passam a ser cobertos por esses tipos de problemas.

As maiores dificuldades são encontradas com relação aos tópicos 8, 10 e 11 do componente curricular "Arquitetura de Computadores Avançada". Em parte, essa dificuldade se explica pelo fato dos alunos estarem iniciando o curso de Engenharia de Computação, pois o estudo integrado de Sistemas Digitais ocorre no segundo semestre do primeiro ano de universidade, e por isso, os alunos ainda não possuem uma base muito sólida, seja na área de *hardware*, seja na área de *software* e estrutura de dados para a realização de projetos deste nível. No entanto, a principal barreira com relação aos tópicos de arquiteturas superescalares, VLIW e sistemas multiprocessados está relacionada ao pouco tempo que os alunos dispõem para a execução dos projetos. Projetos que envolvam a implementação em *hardware* de uma estrutura de dados relativamente complexa para a realização da alocação dinâmica, previsão de desvios e execução especulativa, seja através de registradores lógicos/físicos seja através do *buffer* de reordenamento, passa a ser um obstáculo que tomaria um tempo muito grande dos alunos, o que impossibilitaria a aplicação de outros problemas e reduziria o grau de cobertura dos tópicos do componente PBL.

Vale ressaltar que o conjunto de problemas aplicados no componente curricular PBL tem por objetivo principal introduzir, estruturar e aprofundar os conteúdos visto, na teoria, nos componentes curriculares "Arquitetura de Computadores" e "Arquitetura de Computadores Avançada".

2.3.2. Elaboração de Problemas/Projetos

Um dos aspectos mais importantes para o sucesso da metodologia PBL está relacionado à elaboração de problemas/projetos. Podemos destacar cinco características relevantes para a definição de um bom problema (Santos *et al.*, 2008a) (Angelo & Santos, 2009):

- Um problema deve motivar os estudantes. Inserir elementos próximos da realidade dos estudantes é uma estratégia usada para tornar os problemas mais atraentes e motivar os estudantes na busca de soluções.
- Os problemas devem levar os estudantes a tomar decisões ou realizar julgamentos, baseados em fatos, informações e/ou argumentações lógicas. Os problemas devem conduzir os estudantes para a construção dos argumentos e busca de informações. Isso não quer dizer que todas as informações do problema devem ser relevantes para sua solução. Além disso, alguns problemas podem ser projetados para fornecer informações em diferentes etapas durante a sua solução.
- Os problemas devem ser complexos o bastante para que seja necessária a cooperação de todos os membros em sua solução. A complexidade deve ser pensada de forma que seja possível adotar a estratégia de "dividir para conquistar", ou seja, separar o problema em partes mais simples para simplificar sua solução.
- As questões iniciais dos problemas devem ser abertas, baseadas em conhecimentos prévios e/ou controversas, de forma a proporcionar discussão entre os alunos. Este aspecto desperta o sentimento de grupo entre os estudantes.
- Os objetivos de aprendizagem devem ser incorporados ao problema. Alguns autores propõem que estes objetivos sejam apresentados somente depois da solução. No entanto, nos estudos integrados de Sistemas Digitais, os objetivos de aprendizagem costumam ser incorporados ao problema.

A elaboração dos problemas é um processo colaborativo entre todos os tutores que participam do estudo integrado. Para cada problema define-se um tema, os objetivos de aprendizagem, uma descrição do problema, um cronograma dos tutoriais e aulas, bem como os prazos para a entrega da solução, especificação do produto a ser entregue e um detalhamento de recursos de aprendizagem. Os problemas neste estudo integrado envolvem, na maioria dos casos, a execução de projetos com desenvolvimento de produtos, sejam relatórios, programas, ou soluções utilizando dispositivos lógicos programáveis, como FPGAs.

2.3.3. Avaliação

As avaliações dos estudos integrados de engenharia de computação da UEFS são compostas por um conjunto de elementos. Nos estudos integrados de Sistemas Digitais, bem como em outros estudos integrados da UEFS, como Programação, Concorrência e Conectividade, as avaliações são baseadas no produto elaborado a partir dos problemas e no desempenho do aluno durante as sessões tutoriais (Oliveira *et al.*, 2007) (Santos *et al.*, 2008a). A avaliação do produto é composta pela análise de um programa desenvolvido

e/ou um relatório com resultados e conceitos teóricos referentes ao problema, além de uma sessão denominada "bate-bola". Para a realização desta sessão, um barema de avaliação é definido em função dos objetivos que devam ser atendidos em cada problema. Os baremas são utilizados com o intuito de definir um critério único para ser utilizado por todos os tutores durante a avaliação dos produtos. Na Tabela 2.1 apresenta-se um barema genérico, o qual pode ser adaptado dependendo do problema que está sendo aplicado.

Tabela 2.1. Padronização no processo de avaliação do produto pelos tutores.

Item	Aspecto	Pontuação
Formatação e Estilo	Organização do texto, normas ABNT	0,5
	Gramática e Ortografia	0,5
Conteúdo Teórico	Introdução coerente e motivadora	0,5
	Conceitos teóricos corretos e adequados	1,0
	Referências apropriadas e citadas no texto	0,5
Solução	Resultado e detalhamento da solução	6,5
	Conclusões sobre o projeto	0,5

Como pode ser observado na Tabela 2.1, para a formatação/estilo do relatório está sendo considerado um ponto, para o conteúdo teórico atribuíram-se dois pontos e para a solução do problema, sete pontos. Penalizações também podem ser aplicadas nos casos de atraso, ausência de cópia impressa e cópias de textos entre os alunos. Com a soma destes itens tem-se a nota do produto.

No bate-bola, o tutor discute individualmente sobre seu produto com o aluno, que apresenta sua solução e demonstra conhecimentos adquiridos. O tutor diagnostica o aprendizado, aponta deficiências no aprendizado e esclarece pontos que não foram tratados durante os encontros tutoriais, realimentando o processo de aprendizagem. Neste momento o tutor deve fazer uma avaliação, aplicando um conceito (BB) que varia de 0 até -5, conforme apresentado na Tabela 2.2. O conceito 0 é aplicado ao aluno que demonstra ter adquirido todos os conhecimentos de acordo com os objetivos que deveriam ser atingidos pelo problema, e -5 para o aluno que não conseguir atingir nenhum dos conhecimentos propostos com o problema.

Tabela 2.2. Composição de notas.

Elementos	Pontuação
Produto	10,0
Bate-Bola(BB)	de -5,0 até 0
Produto Corrigido(PC)	10,0
Nota final do produto(PC - BB)	10,0
Desempenho nas sessões tutoriais	10,0

Em geral, depois da avaliação do bate-bola, os alunos têm oportunidade de corrigir seu produto (PC) demonstrando o aprendizado dos pontos falhos apontados pelo tutor, aprofundando a discussão e compreensão destes, e melhorando a avaliação dos produtos elaborados.

Na tabela 2.3, mostra-se os principais pontos que devem ser observados pelo tutor no processo de atribuição de nota referente à solução ou produto obtido pelos alunos. Note-se que, as pontuações referenciadas nas tabelas 2.1, 2.2 e 2.3 indicam apenas uma referência base e não são absolutas para todos os problemas propostos, podendo ser adaptadas de acordo com as características particulares e necessidades de cada problema ou tópico abordado.

Tabela 2.3. Resultado e detalhamento da solução.

Pontos a serem observados	Pontuação
Explicação e nível de detalhamento adequado do produto	2,5
Desempenho e custo do produto obtido	2,0
Comparações com outras soluções	1,0
Identificação de limites, apresentado vantagens e desvantagens da solução alcançada	1,0

Além das avaliações dos produtos, uma nota de desempenho complementa o processo de avaliação do aluno no estudo integrado. Esta nota reflete a participação, o envolvimento, o comprometimento nos encontros do grupo tutorial, as contribuições levantadas, o cumprimento das metas definidas em cada encontro, a organização dos trabalhos do grupo, entre outras habilidades relacionadas à dinâmica em grupo. Além dessa forma de avaliação do desempenho do aluno, existe também a possibilidade da auto-avaliação. Neste procedimento, os tutores, inicialmente, fazem uma avaliação de cada aluno ao final de cada encontro da sessão tutorial, justificando a nota com base nos objetivos de desempenho. Depois de alguns encontros e familiarização dos alunos com os critérios de avaliação, inicia-se um processo de auto-avaliação de cada aluno, que deve atribuir sua nota, justificando-a para todos os alunos do grupo.

Quanto aos componentes curriculares convencionais, ou seja, os módulos de "Arquitetura de Computadores" e "Arquitetura de Computadores Avançada" são realizadas avaliações pelo professor responsável pelas duas disciplinas. Normalmente, as avaliações dos dois componentes convencionais são provas teóricas escritas, aplicadas duas em cada módulo. No entanto, na avaliação final do módulo, pode ser considera ou não uma ou mais avaliações referentes aos produtos produzidos no componente PBL.

Para a composição da média final dos alunos, tem sido adotada uma média aritmética que pode variar dependendo do nível de dificuldade cobrado tanto nos componentes convencionais como no componente PBL. Atualmente, os dois componentes convencionais respondem a 70% da nota final do aluno no estudo integrado, enquanto o componente PBL é responsável pelo restante, 30%. Neste caso, vale destacar a importância da avaliação do produto, que pode entrar na composição tanto dos componentes convencionais, como do estudo integrado, já que os produtos refletem os conteúdos de aprendizagem de forma integradora.

2.4. Estudo de Casos

Nesta seção serão analisados problemas aplicados no Estudo Integrado de Sistemas Digitais, os quais foram divididos em quatro categorias: Projeto de um Processador; Sis-

temas de Memória; Mecanismos de Entrada e Saída; e Seminários. Cada caso engloba três exemplos de problemas, cada qual acompanhado pela descrição sucinta do produto esperado. Ao final de um estudo de caso, são detalhados os conceitos explorados pelo conjunto de problemas, assim como as dificuldades encontradas em seu desenvolvimento e os resultados obtidos pelos alunos.

2.4.1. Caso 1: Projeto de um Processador

Este caso abrange problemas referentes ao projeto de um processador, os quais são relativamente complexos para um aluno de segundo semestre, uma vez que exigem o estudo e a aplicação de conceitos não vistos (ou pouco abordados) até então, envolvendo aspectos arquiteturais, linguagens de descrição de *hardware* e ferramentas de EDA.

2.4.1.1. Exemplo 1: Processador Seqüencial

Este problema apresentou diferentes versões, de forma a ser aplicado a turmas distintas. Em sua primeira versão, foi solicitada a implementação de um processador em FPGA, usando VHDL, com as seguintes características: arquitetura microprogramada; capacidade de endereçamento de 64 kbytes de memória principal; operações em 8 bits; 4 registradores de propósito geral, sendo um deles o acumulador; conjunto de instruções (Tabela 2.4), modos de endereçamento e formatos de instruções fornecidos (Figura 2.3).

Modo de endereçamento imediato (*immediate addressing mode*)
operando = *constante* = *immediate value*

Campo	Opcode (MOVC)		Destination Register		ImmediateValue	
Tamanho	5		3		8	
Faixa Bits	15	11	10	8	7	0

Modo de endereçamento direto por registrador (*register direct addressing mode*)
operando = *conteúdo do registrador*

Modo de endereçamento indireto por registrador (*register indirect addressing mode*)
operando = *conteúdo da posição de memória especificada pelo conteúdo do registrador*

Campo	Opcode		AddrMode (00|01|10)*		reserved		Destination Register		Source Register	
Tamanho	5		2		3		3		3	
Faixa Bits	15	11	10	9	8	6	5	3	2	0

* *bit 10 = modo de endereçamento do registrador destino (DestinationRegister) (0=direto; 1=indireto)*
bit 9 = modo de endereçamento do registrador fonte (SourceRegister) (0=direto; 1=indireto)

Figura 2.3. Formatos de instruções.

A segunda versão do problema alterou o conjunto de instruções, o tipo de arquitetura (*LOAD-STORE*), a capacidade de endereçamento da memória principal (4 kbytes), o tamanho dos operandos (16 bits) e omitiu o formato de instruções. A terceira versão promoveu alterações semelhantes, além de acrescentar um *prefetch* buffer para o armazenamento de 8 instruções.

Tabela 2.4. Conjunto de instruções para o processador (versão 1).

Tipo	Instrução	Operandos	Descrição
Transferência de dados	MOVC	$R, Const$	$R \leftarrow Const$
	MOV	R_1, R_2	$R_1 \leftarrow (R_2) \mid R_1 \leftarrow R_2 \mid (R_1) \leftarrow R_2$
Aritmética	ADD	R_1, R_2	$R_1 \leftarrow R_1 + R_2$
	SUB	R_1, R_2	$R_1 \leftarrow R_1 - R_2$
	MUL	R_1, R_2	$R_1 \leftarrow R_1 * R_2$
	DIV	R_1, R_2	$R_1 \leftarrow R_1 / R_2$
	NEG	R	$R \leftarrow -R$
	ABS	R	$R \leftarrow \lvert R \rvert$
	INC	R	$R \leftarrow R + 1$
	DEC	R	$R \leftarrow R - 1$
Lógica	AND	R_1, R_2	$R_1 \leftarrow R_1$ AND R_2
	OR	R_1, R_2	$R_1 \leftarrow R_1$ OR R_2
	NOT	R	$R \leftarrow$ NOT R
	CMP	R_1, R_2	compara R_1 e R_2, configurando *flags*
Transferência de controle	BR	R	desvio incondicional para o endereço R
	BRC	R	desvio para o endereço R, se $carry = 1$
	BRZ	R	desvio para o endereço R, se $zero = 1$
	BRO	R	desvio para o endereço R, se $overflow = 1$
	BRS	R	desvio para o endereço R, se $sign = 1$
	NOP	-	nenhuma operação

2.4.1.2. Exemplo 2: Processador com Pipeline

Para não esgotar o formato do processor seqüencial, este problema solicitou a implementação em FPGA, através de VHDL, de uma arquitetura paralela com as seguintes especificações: presença de 5 estágios funcionais de execução paralela; interfaceamento com módulos de memória separados para dados e instruções, cada qual com capacidade de armazenamento de 4 Gbytes e endereçamento por bytes (modo *Big Endian*); instruções e operações internas em 32 bits; 16 registradores de propósito geral (R0-R15); conjunto de instruções fornecido; modos de endereçamento imediato, base-deslocamento e a registrador; possibilidade de lidar com 3 *hazards RAW* consecutivos, sem ter que paralisar a propagação de instruções entre os estágios funcionais.

2.4.1.3. Exemplo 3: Processador Digital de Sinais

Este problema buscou a compreensão do funcionamento de uma arquitetura para um sistema transformacional, através do projeto e implementação de um módulo IDCT (*Inverse Discrete Cosine Transform*) em FPGA, voltado para o padrão MPEG-2 MP@ML, usando a linguagem Verilog.

Após breve explicação do Sistema Visual Humano, o texto do problema apresentou expressões das transformadas discretas de cosseno direta e inversa, relacionando amostras dos componentes de brilho e de cor de um bloco de 8x8 pixels, no domínio

espacial, com coeficientes da transformada, no domínio das freqüências. A partir deste cenário, foi solicitado aos alunos a implementação de um sistema transformacional, capaz de decodificar um quadro completo de imagem, dentro das restrições temporais do padrão, considerando-se a sub-amostragem 4:2:0.

2.4.1.4. Conceitos Trabalhados

De forma geral, os problemas do Caso 1 trabalharam com os conceitos expressos na Tabela 2.2, com graus variados de aprofundamento do escopo tratado.

2.4.1.5. Dificuldades no Desenvolvimento

As dificuldades de desenvolvimento dos problemas de Projeto de um Processador variaram de acordo com a turma e o problema, mas, em geral, houve certa uniformidade entre grupos tutoriais distintos, ao tratarem de um mesmo problema. Na seqüencia, as principais dificuldades são descritas, relacionando-as com os exemplos tratados.

Aritmética binária: Nos problemas dos exemplos 1 e 2, ao projetarem a ULA, alguns alunos cometeram pequenas falhas, relativas à representação de números negativos (bit de sinal/complemento de 2). Como a aritmética binária é dada em semestre anterior, no Estudo Integrado (EI) de Introdução ao *Hardware*, solicitou-se maior ênfase sobre o seu ensino.

Portas lógicas: Na maioria dos problemas deste caso, foi solicitada a implementação de parte do processador através de portas lógicas, para que os alunos não perdessem este tipo de abstração. Pode-se observar que melhores resultados de aprendizagem foram obtidos ao solicitar-se a representação da ULA através de portas lógicas, assim como dos registradores através de *flip-flops*. Já a UC demandou um tempo de projeto maior, comprometendo o resultado final, devido à extensão da representação.

Máquinas de estados finitas: A modelagem de tais máquinas mostrou-se um ponto crítico no desenvolvimento dos projetos. Assim, solicitou-se maior ênfase sobre a modelagem e implementação dessas máquinas no EI de Introdução ao *Hardware*.

Ferramentas: Nos primeiros semestres letivos em que o EI foi ministrado, os alunos desconheciam o ambiente utilizado (*Altera Quartus II*), o que implicou em um aumento considerável de aulas de consultoria. Para suavizar a curva de aprendizado do ambiente, o EI de Introdução ao *Hardware* passou a adotá-lo em parte de seus problemas, explorando, somente, a representação em nível de portas lógicas.

Linguagens de descrição de *hardware*: Nas versões 1 e 2 do exemplo 1, os alunos desconheciam totalmente a linguagem VHDL e seus paradigmas, o que provocou um aumento do número estimado de aulas de consultoria (de duas para quatro). A partir da versão 3 do exemplo 1 (e nos demais exemplos), procurou-se amenizar tal dificuldade, através da inserção de aulas teóricas de VHDL (e, posteriormente,

Tabela 2.5. Conceitos trabalhados nos problemas dos caso 1.

Conceito	Escopo Tratado	Exemplo
Componentes internos à UCP	ULA (Unidade Lógica-Aritmética) UC (Unidade de Controle) Registradores	1, 2 e 3
Componentes externos à UCP	Comunicação processador-memória	1, 2 e 3
ULA	Aritmética binária Configuração de *flags*	1, 2 e 3
UC	Microprogramada (uso de microinstruções) versus Hardwired (uso de circuito lógico) Máquina de estados finita Centralizada versus Distribuída	1 e 2 1, 2 e 3 2
Registradores	Propósito específico versus propósito geral Implementação através de circuitos lógicos (proibido o uso de variáveis VHDL/Verilog) Uso de barramento interno único entre registradores: exploração de *buffers tri-state*	1, 2 e 3
Conjunto de instruções	Aritméticas, lógicas, transferência de controle e transferência de dados Máquinas de 1, 2 e 3 endereços Arquiteturas *LOAD-STORE*	1 e 2
Modos de endereçamento	Imediato, direto por registrador e indireto por registrador	1 e 2
Formatos de instrução	Relacionamento com o conjunto de instruções e os modos de endereçamento	1 e 2
Microprogramação	Temporização das instruções: projeto do conjunto de microinstruções e definição de seu sequenciamento (programa de controle)	1 e 2
Paralelismo	Uso de *pipeline* para explorar ILP (*Instruction-Level Parallelism*)	2
Projeto RTL	Linguagens de descrição de *hardware* (VHDL/Verilog) Ferramentas de síntese e simulação Compromissos conflitantes (*tradeoffs*) entre velocidade de processamento e área ocupada no *hardware*	1, 2 e 3 3

Verilog) com relativa antecedência. Contudo, isto funciona melhor quando o Projeto de um Processador corresponde ao segundo problema dentro do EI, o que dá maior margem de tempo, visto que o aprendizado de uma linguagem de descrição de *hardware* é relativamente complexo (neste ponto do curso, os alunos têm pouco conhecimento até sobre linguagens de programação, tais como C e Java).

Temporização: Uma das principais dificuldades dos alunos consistiu em misturar blocos

combinacionais e seqüenciais em um mesmo projeto, não atentando para a inserção de registradores nas entradas e/ou saídas de determinados blocos combinacionais, além da não observância do tempo de estabilização de tais blocos. Foi comum a adoção de um sinal de sincronismo de período empírico, o qual muitas vezes gerou falhas, devido à etapa de *place & route*, a qual geralmente altera a disposição dos elementos no dispositivo FPGA e, consequentemente, os tempos de propagação de sinais. A falta de domínio das ferramentas de análise temporal, existentes no ambiente utilizado, prejudicou os alunos em suas decisões.

Otimização do programa de controle: Nos problemas dos exemplos 1 e 2, foi facultado aos alunos a escolha da forma de implementação da UC: *hardwired* (projeto baseado na representação estrutural em nível de portas lógicas) ou microprogramada (projeto baseado na representação comportamental em nível RTL). Devido à maior facilidade de projeto, os alunos optaram pela microprogramação. Além da questão de determinar quais sinais de controle a UC ativava simultaneamente (definindo a chamada palavra de controle), uma das dificuldades recorrentes foi a otimização do programa de controle, isto é, a diminuição do tamanho do microprograma. Isto decorreu do baixo (ou inexistente) reuso de código. Exemplo de erro deste tipo foi a geração de trechos repetidos de microprograma para uma instrução `MOV Ri ,Rj`, para todas as combinações possíveis de registradores - tal desperdício de espaço de armazenamento poderia ter sido evitado, através da adoção de um único trecho de microprograma, com o apoio de um circuito decodificador externo, o qual determinasse os operandos fonte e destino, armazenando-os diretamente no registrador de controle (armazenador da palavra de controle).

Interação com a memória: Para simplificar o desenvolvimento dos produtos requiridos, foi permitido aos alunos o uso de *soft IP-cores* para o controle de memórias embutidas no dispositivo FPGA utilizado (ao invés de obrigá-los a desenvolver tais controladores). Apesar disso, em todos os problemas, eles tiveram dificuldades quanto à temporização da memória e os protocolos de comunicação utilizados. Aulas de consultaria foram dadas para resolver esta dificuldade.

Outras dificuldades: Além das dificuldades citadas, outras foram encontradas, de acordo com o problema. No exemplo 1, pode-se destacar a compreensão do ciclo de instrução de um processador e estruturas de armazenamento auxiliares (tipo pilha e fila). No exemplo 2, os conceitos de paralelismo em nível de instruções, *hazards* e UC distribuída por estágios *pipeline*. No exemplo 3, a decomposição de operações matriciais. Para todos os exemplos, o paralelismo de processos em linguagens de descrição de *hardware*. Aulas de consultoria abordaram tais dificuldades.

2.4.1.6. Resultados

Os alunos tiveram bom desempenho no desenvolvimento dos projetos dos exemplos 1 e 2, e baixo desempenho no problema do exemplo 3.

As três versões do exemplo 1 apresentaram implementações contemplando entre 70% e 90% do requerido, de acordo com o grupo tutorial. O problema do exemplo 2

foi totalmente contemplado por um dos grupos tutoriais, atingindo entre 70% e 80% do exigido nos demais grupos - o principal motivo de sucesso do grupo que o implementou em sua totalidade foi o de seguir a arquitetura proposta por Patterson e Hennessy (Hennessy & Patterson, 2008). Já o problema do exemplo 3 foi aplicado, com modificações, em duas oportunidades, ficando a implementação aquém do esperado. Na primeira vez em que foi aplicado, os tutores pouco interferiram na condução das discussões, o que fez com que muitos distratores (presentes no texto do problema) desviassem a atenção do foco do problema - em um dos grupos, os alunos perderam muito tempo discutindo aspectos do esquema de decodificação de audio e vídeo do padrão MPEG-2. Mesmo tendo reformulado o texto do problema, apresentando-o para nova turma de forma mais direta, os alunos não foram bem sucedidos em sua implementação, devido, sobretudo, à complexidade matemática da simplificação de operações matriciais com componentes cosseno.

2.4.2. Caso 2: Sistemas de Memória

Neste caso serão apresentados três problemas que abordam tópicos relacionados ao controle, gerenciamento e acesso à sistemas de memória. Dois problemas tem por objetivo o desenvolvimento de projetos de circuitos digitais, utilizando *softwares* para simulação e síntese, enquanto que um deles utiliza a linguagem *assembly* como ferramenta.

2.4.2.1. Exemplo 1: Memória Cache e Hierarquia de Memória

O objetivo desse problema foi a implementação de um controlador de memória cache em FPGA, usando a linguagem VHDL, capaz de se adaptar a um processador e uma memória principal com as características listadas na tabela 2.6.

Tabela 2.6. Caracteristicas da arquitetura do Exemplo 1 do Caso 2.

Processador	Memória Principal	Cache
10 bits de endereçamento	SDRAM	capacidade de 64 bytes
barramento de dados de 8 bits	capacidade de 1k x 8 (estipulada para o problema)	mapeamento associativo por conjuntos, *2-Way*
clock de 10MHz	linha de dados de 64 bits	4 linhas por conjunto
sinais de controle: R (ler da memória), W (escrever na memória) e ACK (sinal de entrada, que indica o fim uma operação realizada)	sinais de controle: R (leitura), W (escrita), *BURST* (modo rajada, 6-1-1-1) e CLK (*clock*)	linhas de 8 palavras, sendo cada palavra 1 byte
		política de substituição: FIFO (*First In First Out*) ou LRU (*Least Recently Used*)

2.4.2.2. Exemplo 2: Memória *Cache* / Linguagem *Assembly*

O problema deste exemplo apresentava para os alunos o seguinte "desafio": comprovar a existência de memória cache nos computadores do laboratório de *hardware* da UEFS (cujas máquinas na época possuíam processadores Pentium 4). Para isso um programa de teste em *assembly* deveria ser construído.

2.4.2.3. Exemplo 3: Esquemas de Endereçamento

A motivação do exemplo 3 foi a necessidade que os sistemas multitarefa tem de desvincular os endereços das instruções e dados, relativos a um determinado programa, de seus endereços absolutos. O uso de esquemas de segmentação e paginação permitem ao sistema operacional liberdade de alocar partes de um programa em execução em regiões de memória que melhor lhe convier, de acordo com o conjunto de programas em execução, num dado momento.

Foi apresentado no texto do problema o esquema de segmentação e paginação definido na arquitetura de 32 bits da Intel (IA-32), conforme as Figuras 2.4 e 2.5.

A partir dessa arquitetura foi solicitado aos alunos que projetassem e implementassem os módulos referentes ao esquema de tradução de endereços virtuais em físicos da arquitetura IA-32. Os módulos deveriam ser codificados em VHDL e a implementação ser feita no *software* de EDA *Quartus II*, da *Altera*.

2.4.2.4. Conceitos Trabalhados

De forma geral, os problemas do Caso 2 trabalharam com os conceitos expressos na Tabela 2.7.

2.4.2.5. Dificuldades no Desenvolvimento

Os principais problemas encontrados nos problemas do Caso 2 são enumerados a seguir.

Compreensão dos conceitos envolvidos: Uma vez que o texto do exemplo 2 não apresentava nenhuma "pista"sobre o assunto, nas primeiras sessões as discussões teóricas foram pouco produtivas, uma vez que os alunos precisaram de algum tempo para compreenderem os conceitos envolvidos. No caso do exemplo 3 a maior dificuldade foi a compreensão do mecanismo de conversão de endereços.

Estabelecimento de critérios de teste: No problema A busca por uma estratégia que permitisse verificar a existência da memória cache só pode ser atacada a partir da compreensão dos mecanismos envolvidos no seu gerenciamento. Uma das dificuldades encontradas foi como estabelecer critérios para testar os limites da cache (tamanho de linha / número de linhas por conjunto). A abordagem adotada por todos os grupos tutoriais foi a elaboração de vetores de teste, de forma a provocar substituição de linhas na *cache*.

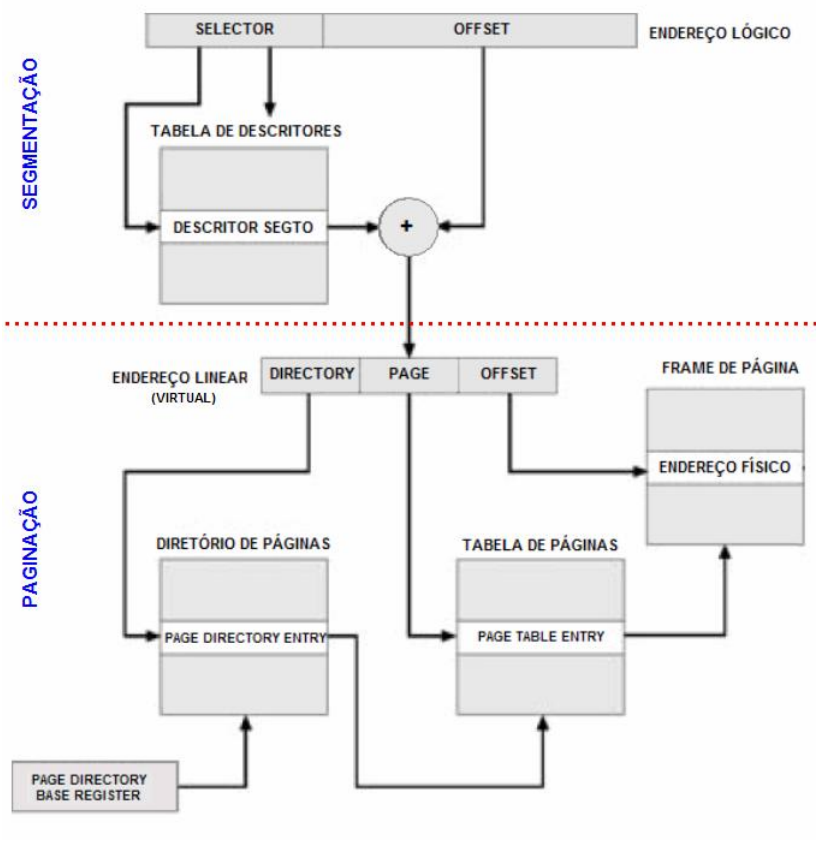


Figura 2.4. Esquema de segmentação paginada na arquitetura IA-32.

Linguagem *Assembly*: Esse problema foi o primeiro contato dos alunos com a linguagem *Assembly*. Além da dificuldade em assimilar suas características, o uso de interrupções em *Assembly* também se mostrou um ponto crítico no processo de aprendizado.

Uso da ferramenta de auxílio: No texto do problema era indicado como ferramenta de auxilio a ferramenta *emu8086*. O curva de aprendizado para essa ferramenta foi mais alta do que o esperado.

Eliminação de efeitos colaterais do sistema operacional: Os alunos demoraram a perceber que o sistema operacional poderia exercer influência nos resultados obtidos, dependendo da estratégia adotada para provocar falhas na *cache*.

Dificuldades de implementação: Durante a fase de implementação do exemplo 1 os alunos tiveram dificuldades em atacar a política de substituição LRU. Em determinados casos, a forma simplificada da implementação falhava em relação à escolha da linha menos recentemente usada. Foi adotada para a política de atualização a opção de forma mais simples: *write-through*. As implementações da falta de escrita foram tratadas apenas na memória principal. No exemplo 3, os alunos não conseguiram implementar o diretório de *cache* e o diretório e a tabela de páginas.

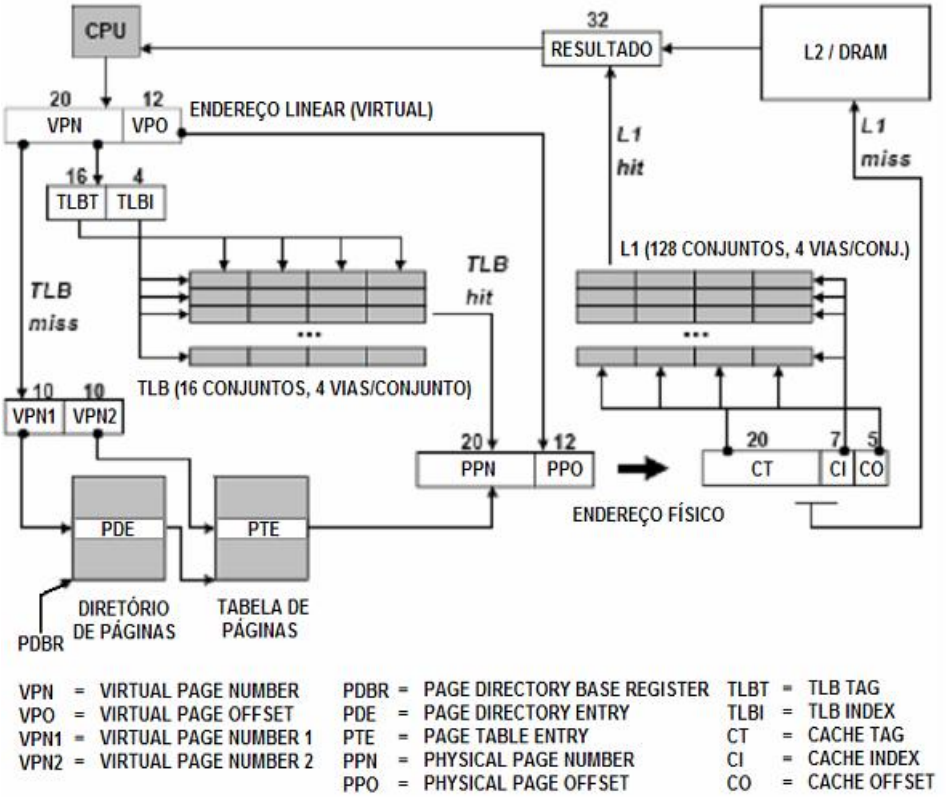


Figura 2.5. Detalhes do esquema de paginação (parte inferior da Figura 2.4).

2.4.2.6. Resultados

O desempenho alcançado pelos alunos nos exemplos 1 e 2 foi muito bom, com um aproveitamento de 80% e 95%, respectivamente. Já no exemplo 3 o desempenho foi fraco, com aproveitamento em torno de 40%.

No exemplo 1 a única falha cometida pelos alunos foi na implementação da política de substituição de linhas LRU, na qual ocorriam situações de mal funcionamento, não removendo a linha menos recentemente utilizada. Contudo, tal situação foi explicitada nos relatórios apresentados.

O exemplo 2 foi bastante interessante em suas primeiras discussões, que envolviam questões teóricas interessantes sobre o funcionamento das memórias *cache*, seus níveis e estratégias de gerenciamento. No entanto, a partir do momento que uma proposta de algoritmo foi definida, as sessões tutoriais passaram a ser pouco produtivas, uma vez que os problemas enfrentados pelos grupos eram fundamentalmente técnicos, vinculados a implementação na linguagem *assembly*.

No exemplo 3 a dificuldade de compreensão dos conceitos envolvidos e a complexidade do problema foram fatores determinantes para o mal desempenho obtido.

Tabela 2.7. Conceitos trabalhados nos problemas dos caso 2.

Conceito	Escopo Tratado	Exemplo
Organização de memória	Hierarquia de memória	1 e 2
	Endereços lógicos, lineares e físicos	3
	Esquemas de tradução de endereços em ambientes multitarefa na arquitetura IA-32	
	Esquemas de segmentação e paginação para controle de memória virtual	
	Tradução de endereços virtuais em físicos	
Memória *cache*	Princípios de localidade (temporal / espacial)	1 e 2
	Organizações de memória *cache* (completamente associativa / conjunto-associativa / mapeamento direto)	
	Tratamento de faltas (leitura / escrita)	
	Políticas de substituição de linhas (FIFO / LRU)	
	Políticas de atualização (*write-through* / *write-back*)	
Projeto RTL	Simulação de circuitos digitais	1 e 3
	Síntese de circuitos digitais utilizando VHDL	
Outros	Arquitetura 80x86	2 e 3
	Linguagem *assembly* (80x86)	
	Programas de teste	2

2.4.3. Caso 3: Mecanismos de Entrada/Saída (E/S)

Os problemas discutidos nesta seção tratam de noções básicas sobre barramentos e dispositivos de E/S, visando que os alunos compreendam o funcionamento de barramentos, arbitragem de barramentos, mecanismos de acesso ao barramento, tipos de barramentos e os princípios básicos de entrada e saída de dados para o processador. Serão apresentados três exemplos de problemas resolvidos pelos alunos, o primeiro deles se refere a uma arquitetura multiprocessada que compartilha uma memória de dados. O segundo exemplo é o desenvolvimento de um árbitro de barramento. O terceiro mostra o projeto de um controlador para barramento PCI (*Peripheral Component Interconnect*).

2.4.3.1. Exemplo 1: Barramento de Memória para uma Arquitetura com Múltiplos Processadores

Um dos produtos relacionados ao tema barramento e dispositivos de E/S, se refere ao desenvolvimento de um controlador para o barramento de dados de uma arquitetura multiprocessada, implementado em FPGA, utilizando a linguagem VHDL. A arquitetura é constituída por quatro processadores, cada um deles possui 32kB de memória de dados local e 4kB de memória de programas local. O processador em questão, foi desenvolvido pelos alunos em um problema anterior. Os quatro processadores compartilham 128kB de memória de dados.

2.4.3.2. Exemplo 2: Controle de Acesso a um Barramento

O segundo problema apresentado sobre este tema, se refere ao projeto de um árbitro de barramento utilizado por uma CPU para ter acesso a quatro dispositivos de E/S. A arquitetura era composta por uma CPU, uma memória e quatro dispositivos de E/S conectados por meio de um barramento, controlado por um árbitro de barramento com gerenciamento adaptativo de prioridades. Cada dispositivo de E/S transfere uma quantidade distinta de palavras, e foram classificados da seguinte forma:

- dispositivo A, que transfere palavras isoladas;
- dispositivo B, que transfere blocos de quatro palavras;
- dispositivo C, que transfere blocos de oito palavras;
- dispositivo D, que transfere blocos de tamanho variável, contendo de uma a dez palavras.

Outra restrição de projeto era a forma de acesso ao barramento quando eram realizadas requisições simultâneas. Se dois dispositivos solicitassem o barramento ao mesmo tempo para transferir até oito palavras, a prioridade de ambos seria a mesma e o árbitro obedecia a política *round-robin*. Caso um dos dispositivos solicitasse uma transmissão superior a oito palavras, este dispositivo teria acesso ao barramento para transmitir apenas quatro palavras, em seguida o outro dispositivo enviaria seu bloco de palavras, e o acesso retornaria ao primeiro dispositivo para que o restante das palavras fossem transferidas.

O árbitro do barramento foi implementado em uma FPGA utilizando a linguagem VHDL, onde a CPU era apenas um processo que disponibilizava, periodicamente, o barramento e uma memória, de tamanho máximo de 6 KBytes, foi preenchida somente com valores de 0 a 9 para simular os dados a serem enviados pelos dispositivos. As solicitações do barramento pelos dispositivos foram feitas por meio de chaves do tipo *push-button*. Um LED indicava a disponibilidade do barramento. Quatro LED's, um para cada dispositivo, indicavam qual estava usando o barramento. Para verificar se o dado transferido estava correto foi utilizado um *display* de 7-segmentos.

2.4.3.3. Exemplo 3: Controlador de Barramento PCI

O terceiro exemplo sobre barramentos e dispositivos E/S foi inspirado na idéia de desenvolvimento de uma placa de vídeo simples que pudesse ser ligada a um processador desenvolvido em um problema anterior. Visando o entendimento de como essa conexão é feita em dispositivos disponíveis no mercado, foi escolhido o barramento PCI. Para que o dispositivo pudesse ser construído em laboratório, devido as restrições de memória da FPGA disponível, foi adotado um padrão entrelaçado de exibição de imagens em televisores com entrada vídeo componente, compatível com o formato PAL 4:2:0.

O problema consistiu em projetar, e implementar em FPGA um controlador de barramento PCI, que permita a conexão de até 4 controladores de E/S ao barramento.

Também era parte do projeto o desenvolvimento do *hardware* para o interfaceamento de dois controladores de E/S, um deles para um placa responsável pela geração de amostras digitais dos sinais de vídeo componente que seriam gravadas na memória principal, e um para o *hardware* de exibição de imagens, equipado com *buffer* que permita o armazenamento de uma linha horizontal de imagem, proveniente da memória principal. Ambos controladores de E/S deviam possuir mecanismos DMA (*Direct Memory Access*) embutidos.

2.4.3.4. Conceitos Trabalhados

Os conceitos trabalhados no problemas do caso 2.4.3, estão apresentados de forma resumida na Tabela 2.8.

Tabela 2.8. Conceitos trabalhados nos problemas do caso 3.

Conceito	Escopo Tratado	Exemplo
Barramentos	Tipos de barramentos (número de fios, tipo de informação transferida, localização física)	1, 2 e 3
	Técnicas de comunicação (síncrona, assíncrona)	
	Arbitragem (centralizada, distribuída)	
	Tratamento de prioridades	2 e 3
	Padrão PCI	3
Dispositivos E/S	Tipos de dispositivos (mestre, escravo)	1, 2 e 3
	Controle de acesso a dispositivos externos por meio de um barramento de dados comum	1
	Controladores de acesso direto à memória (DMAC)	2 e 3
Processamento paralelo	Topologias	1
	Compartilhamento de memória de dados	
Padrões de vídeo	Formato PAL 4:2:0 entrelaçado	3
Projeto RTL	Linguagens de descrição de *hardware* VHDL	1, 2 e 3
	Simulação de circuitos digitais	

2.4.3.5. Dificuldades no Desenvolvimento

A metodologia PBL se caracteriza pela construção do conhecimento pela resolução de problemas. Geralmente, os alunos não tem uma base conceitual forte ao início de um problema, isto gera algumas dificuldades na resolução dos mesmos. Cronologicamente, os casos 1 e 2 são apresentados aos alunos antes dos problemas relacionados aos mecanismos de entrada e saída. Em comparação com os casos anteriores, os problemas com a linguagem de descrição de *hardware* praticamente inexistem e as dificuldades se concentram em como aplicar os conceitos sobre os dispositivos para resolver os problemas.

O problema descrito na seção 2.4.3.1 apresentou os problemas listados a seguir.

Adequação do processador: Para construir a arquitetura pedida no problema, os alunos deveriam adequar o processador desenvolvido adicionando sinais de controle para realização da comunicação entre os processadores. Alguns alunos apresentaram problemas nesta modificação devido a forma com que foi desenvolvida a unidade de controle do processador.

Controle de acesso: Os quatro processadores deveriam acessar uma memória compartilhada utilizando um único barramento, que se faz necessário um controle de acesso referente as requisições de acesso. O mecanismo de controle deveria permitir o acesso igualitário entre os processadores. Este mecanismo de acesso não foi implementado.

Fila de requisições: As requisições de acesso deveriam obedecer uma fila, uma fez que o controle de acesso não foi implementado a fila de requisições não funcionou.

Transferência de pacotes: A transferência entre as memórias locais e a compartilhada deveria ser feita em pacotes, o que não foi alcançado, sendo transmitido apenas um byte por vez intermediado pelo processador.

Elaboração do problema: O enunciado do problema se mostrou vago, resultando em um grande número de elementos distratores, que dificultou a compreensão de pontos fundamentais a implementação do problema. Além disso, o mal dimensionamento do calendário das sessões fez com que as sessões ficassem intercaladas com feriados e um congresso no qual parte dos alunos participou.

O exemplo da seção 2.4.3.2 encontrou as seguintes dificuldades:

Gerenciamento de prioridades: O sistema adaptativo baseado na quantidade de palavras não foi implementado. Alguns alunos não conseguiram realizar a contagem das palavras a serem transferidas, que era necessário para o gerenciamento adaptativo.

***Debouncer*:** As solicitações de acesso ao barramento foram simuladas usando chaves *push-button* acionadas pelos alunos. Estas chaves precisam ser associadas a circuitos digitais denominados *Debouncer*. Estes circuitos funcionam como um tipo de temporizador, uma vez acionada a chave a saída do circuito permanece em nível alto durante um tempo. Isso diminui o efeito de ruídos e garante que possam ser testados os casos de requisições múltiplas. Os alunos não conseguiram implementar este circuito que teve que ser disponibilizado pelos tutores.

A seção 2.4.3.3 trata de um controlador de barramento PCI, foram encontradas as seguintes dificuldades:

Tratamento de prioridade: O arbitro do barramento deveria realizar o tratamento de prioridades quando houvessem requisições múltiplas. Alguns grupos assumiram que essa situação não ocorreria e não implementaram o gerenciamento das mesmas.

Padrão de vídeo: O estudo sobre o padrão de vídeo adotado consumiu um tempo maior que o estimado. A informação a ser extraída do padrão era apenas o tamanho do *buffer* a ser utilizado,. Os alunos se aprofundaram demais no assunto o que diminui o tempo de pesquisa sobre o padrão PCI e de desenvolvimento do *hardware*.

Interface: A construção da interface que seria entregue aos alunos para teste do *hardware* implementado não ficou disponível em tempo hábil para a realização dos testes práticos. Por isso, foi apresentada apenas a simulação do projeto, que ocasionou a uma dificuldade maior na apresentação dos resultados.

2.4.3.6. Resultados

O desenvolvimento do projeto apresentado no problema da seção 2.4.3.1 foi prejudicado por eventos externos. O produto apresentado pelos alunos não atendeu as especificações exigidas, conseguindo apenas a comunicação byte a byte entre dois processadores apenas. As requisições de acesso não obedeceram ao esquema de fila, sendo realizadas de forma independente. Em resumo, se um processador solicitasse o uso do barramento, o mesmo seria obtido independente de que o barramento estivesse sendo utilizado por outro processador. Apesar do não funcionamento, os conhecimentos adquiridos pelos alunos cobriram entre 50 e 60% do assunto abordado neste problema.

O exemplo da seção 2.4.3.2 foi implementado quase por completo, faltando apenas a parte adaptativa do gerenciamento de prioridade. No que se refere aos tópicos abordados, o aproveitamento dos alunos foi entre 70 e 80%. Por fim, o problema da seção 2.4.3.3 alcançou o resultado esperado, apesar de um dos grupos não ter implementado o gerenciamento de requisições múltiplas. Praticamente, todos os pontos escolhidos para o problema foram estudados pelos alunos, resultando em um desempenho médio de 85%.

2.4.4. Caso 4: Seminários

Habilidades de apresentação, necessárias à atuação profissional futura dos alunos, podem ser exploradas em problemas PBL. O presente caso mostra três exemplos de problemas, de curto prazo de desenvolvimento (três semanas), cada qual gerando, como produto final, a apresentação de um seminário envolvendo todos os grupos tutoriais.

2.4.4.1. Exemplo 1: Arquiteturas Embutidas para *Videogames*

O objetivo desse problema foi fazer com que os alunos adquirissem noções básicas sobre arquiteturas embutidas. Como motivação foi proposto um estudo sobre a arquitetura do console de *videogame PlayStation 2* da *Sony*. Além de ser uma plataforma bastante conhecida dos alunos, essa arquitetura possui uma vasta documentação disponível tanto na internet quanto em livros, como (Hennessy & Patterson, 2008).

Após um breve histórico da evolução dos jogos eletrônicos e da contextualização desse mercado, o problema propõe a seguinte situação: a empresa para a qual os alunos trabalham pretende lançar jogos com temas regionais e educacionais e precisa que

a equipe de P&D faça um levantamento de requisitos para entrar neste mercado. Como arquitetura alvo é proposta a arquitetura *PlayStation 2* em função de sua base instalada.

O problema sugere que a equipe elabore um levantamento das características arquiteturais desse console e sua forma de programação, com considerável nível de detalhamento. Foi fornecido um diagrama básico dessa arquitetura, como mostra a Figura 2.6 (Hennessy & Patterson, 2008).

Como produto, foi solicitado que os alunos preparassem um seminário, com no máximo 30 minutos duração. Nesse seminário a equipe além de apresentar os resultados obtidos deveria responder a questões levantadas pela audiência, ou seja, tutores e colegas dos outros grupos tutoriais.

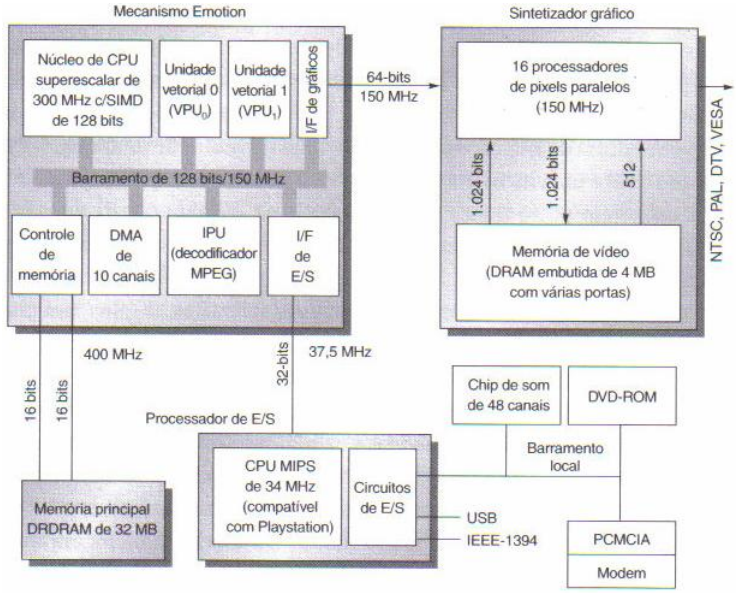


Figura 2.6. Esquema descritivo da arquitetura do *PlayStation 2* - *Sony*.

2.4.4.2. Exemplo 2: Arquiteturas Embutidas para Celulares

A motivação central deste problema era despertar os alunos para a diferença entre as arquiteturas. Uma arquitetura embutida deve lidar com restrições de *hardware* e de alimentação de seus circuitos, devido ao tamanho reduzido do equipamento que a embarca. Visando um exemplo que fizesse parte do convívio diário dos alunos, foi escolhida as arquiteturas de telefonia móvel, para as quais os alunos deveriam compreender o funcionamento básico e como interagem os componentes de *hardware* e de *software* de tais sistemas, assim como a sua relação com o ambiente externo.

Cada grupo ficou encarregado de um estudo aprofundado sobre assuntos de um tópico específico. A separação dos tópicos ficou definida da seguinte forma:

- componentes de *hardware* da arquitetura de um telefone celular e sua interação com os componentes de *software*;

- *software* básico de um telefone celular e sua interação com os recursos de *hardware* do aparelho;
- aplicativos de um telefone celular e sua interação com o *software* básico e com os recursos de *hardware* do aparelho;
- recursos de *hardware* e de *software* de um telefone celular tangentes à comunicação, assim como a rede de interconexão e a arquitetura de estações rádio base.

2.4.4.3. Exemplo 3: *Clusters*

O objetivo deste problema era a compreensão de conceitos inerentes a arquiteturas de multicomputadores, particularmente *clusters*, explorando tanto os aspectos arquiteturais de nós computacionais, quanto as tecnologias de interconexão entre tais nós.

Após uma breve introdução sobre *clusters*, o texto do problema limitou o escopo da pesquisa a ser realizada sobre *commodity clusters*, em especial os montados com GPUs (*Graphics Processing Units*).

Os alunos deveriam encontrar uma GPU, existente no mercado, que servisse de nó computacional. Características arquiteturais da GPU escolhida deveriam ser estudadas e apresentadas, tais como: microarquitetura e interconexão de seus processadores internos; circuitos e dispositivos de E/S; organização de memória; e organização dos dispositivos de armazenamento.

Além das características de uma GPU comercial, os alunos deveriam pesquisar arquiteturas de *clusters* baseadas em tais GPUs, incluindo a rede de interconexão entre nós e a organização de SANs (*Storage Area Networks*). Outros aspectos a pesquisar incluíam: vantagens e desvantagens entre as arquiteturas encontradas; comparação de *clusters* com *mainframes*/supercomputadores especializados; formas de programação; e sistemas operacionais.

2.4.4.4. Conceitos Trabalhados

Os problemas dos exemplos 1, 2 e 3 possuem características bem distintas, de forma que os conceitos trabalhados nos mesmos (e seu grau de profundidade) são apresentados em tabelas separadas - respectivamente, Tabelas 2.9, 2.10, 2.11 e 2.12.

2.4.4.5. Dificuldades no Desenvolvimento

Apesar da diversificação, os problemas propostos como seminários apresentaram dificuldades de desenvolvimento semelhantes, conforme descrição seguinte.

Escopo: Nos três problemas, o texto dava margem a um escopo amplo, cabendo aos alunos a escolha do que seria tratado. Para restringir o escopo, direcionando melhor a pesquisa, no exemplo 1 definimos o que cada grupo tutorial abordaria. Contudo,

Tabela 2.9. Conceitos trabalhados no problema exemplo 1 do Caso 4.

Conceito	Escopo Tratado
Multiprocessamento	Arquitetura da CPU principal (MIPS III)
	Arquitetura da CPU de E/S (MIPS)
	Arquitetura SIMD
	Decodificador MPEG
	Processadores do sintetizador gráfico
	Interação dos processadores
	Formas de programação
Hierarquia de memória	Funcionalidade dos diferentes bancos de memória
	Controle da memória distribuída
	Canais de DMA
Renderização de imagens	*Pipeline* gráfico
	Processadores de vértices e de fragmentos
	Programação baseada em *stream*
Circuitos de E/S	Saída de vídeo (NTSC/PAL/DTV/VESA)
	Controlador USB e IEEE-1394
	Saída de áudio em 48 canais
Outros usos do console	Aplicações científicas
	Uso de sistemas operacionais não nativos (Linux)
	Cluster de consoles

isto restringia muito a compreensão do todo - para amenizar este efeito colateral, nos exemplos 2 e 3 esperamos de uma a duas semanas para comunicar o particionamento entre os grupos.

Pesquisa: No problema do exemplo 1, uma das principais dificuldades dos alunos foi a escassez de referências de qualidade para determinados tópicos, tais como detalhes da arquitetura e formas de programação do *PlayStation 2*. Já nos exemplos 2 e 3, a principal dificuldade dos alunos foi a filtragem de material, amplamente disponível. Em todos os exemplos, a dificuldade de compreensão de termos e conceitos esteve presente.

Nível de profundidade: A dificuldade de pesquisa fez-se acompanhar de um estudo pouco aprofundado de muitos dos conceitos abordados nos problemas. Aspectos como multiprocessamento, organização de memória, redes de interconexão, formas de programação, sistemas operacionais, exploração de paralelismo e protocolos de comunicação foram pouco trabalhados ao longo das sessões tutoriais.

Organização da apresentação: Antes um descuido do que uma dificuldade, a organização da apresentação foi relegada, em todos os problemas, à última sessão tutorial, antes da apresentação.

Tabela 2.10. Conceitos trabalhados no problema exemplo 2 do Caso 4.

Conceito	Escopo Tratado
Hardware Embutido	Processadores, microcontroladores e DSPs
	Memória embutida (tipos de dispositivos)
	Circuitos e dispositivos de E/S
	Cartões de memória
	Acesso e controle de câmeras
	Utilização de circuitos de radio frequência (RF)
	Bootloader
	Codificadores/decodificadores de áudio e vídeo
	Chip da operadora
Sistema Operacional (SO)	Sistemas operacionais para arquiteturas embutidas
	Sistema de arquivos
	Gerenciamento de memória
	Compiladores cruzados/emuladores
Aplicativos	Ferramentas de *software* (livres e comerciais)
	Formas de programação e interação com o SO
	Toolkits gráficos
	Bibliotecas otimizadas em C
	Acionamento de aplicativos por reconhecimento de fala
	Navegadores de Internet
Rede de telefonia móvel	Central de Comutação e Controle
	Células e Cobertura
	Padrões e protocolos de transmissão
	Controle de tráfego
	Moduladores e demoduladores
	Configuração da rede de interconexão
	Canais de dados
	Roteamento
	Arquitetura de estações rádio base

2.4.4.6. Resultados

Os três problemas deste estudo de caso tinham dois objetivos principais: explorar temas não tratados em sala de aula e em outros problemas; e desenvolver habilidades de apresentação. Para propósitos de avaliação, foram considerados os seguintes quesitos (com respectivos pesos): apresentação (20%), organização (20%), tempo (10%), domínio (25%) e clareza de exposição (25%).

Considerando-se os ítens domínio e clareza de exposição, apesar da escassez de referências sobre alguns tópicos, o problema do exemplo 1 (*PlayStation 2*) gerou os melhores resultados, com maior envolvimento por parte dos alunos - uma possível explicação seria a proximidade do tema abordado com o universo jovem. Entretanto, de forma contrária, outro objeto relacionado a tal universo não gerou boas apresentações: o celular (exemplo 2) - diante de extenso material bibliográfico, os alunos produziram apresenta-

Tabela 2.11. Conceitos trabalhados no problema exemplo 3 do Caso 4.

Conceito	Escopo Tratado
Componentes internos à GPU	Arquitetura de multiprocessadores gráficos
	Organização de dispositivos de armazenamento
	Circuitos e dispositivos de E/S
	Organização de memória
	Interconexão de multiprocessadores gráficos/núcleos de processamento/controladores de memória
Arquitetura de multiprocessadores gráficos (*stream processors*)	Núcleos de processamento: *stream processing cores*
	Pipeline gráfico
	Clusters de ULAs
	Unidades de ponto flutuante
	Controladores de memória
Interconexão de multiprocessadores gráficos	Compartilhamento de memória
Interconexão de núcleos de processamento / controladores de memória	Redes de interconexão: *full crossbar*, *mesh* e *ring*
Circuitos e dispositivos de E/S	Canais de acesso à memória
	PCI *Express*
Organização de memória	LRF (*Local Register File*)
	SRF (*Stream Register File*)
	Memória global (*off-chip*)
Organização de dispositivos de armazenamento	Arranjos RAID

ções superficiais, não aprofundando tópicos essenciais, tais como sistemas operacionais, formas de programação e decodificadores embutidos. Por fim, o problema do exemplo 3 produziu apresentações de nível variado, dentro de um mesmo grupo tutorial - devido à extensão do escopo, o seccionamento dos tópicos dentro de cada grupo promoveu uma perda significativa da compreensão geral.

Quanto aos ítens apresentação, organização e tempo, pode-se observar a falta de articulação dos alunos em preparar uma apresentação fechada, coesa, com delineamento adequado de conceitos e terminologia consistente ao longo da apresentação.

2.5. Conclusão

Este capítulo apresentou a experiência do grupo de professores do curso de Engenharia de Computação da Universidade Estadual de Feira de Santana (UEFS), na aplicação da metodologia PBL (*Problem-Based Learning* - Aprendizagem Baseada em Problemas), em disciplinas relacionadas à área de Arquitetura de Computadores, mais especificamente, no estudo integrado de Sistemas Digitais, o qual engloba as disciplinas de Arquitetura de Computadores e Arquitetura de Computadores Avançada.

A experiência relatada faz parte de um processo evolutivo, em andamento, cujas

Tabela 2.12. Conceitos trabalhados no problema exemplo 3 do Caso 4 (cont.).

Conceito	Escopo Tratado
Clusters de GPUs	Comparação com *mainframes*
	Sistemas operacionais
	Formas de programação
	Rede de interconexão entre nós
	Rede de armazenamento (*SAN - Storage Area Network*)
Rede de interconexão entre nós GPU	*Gigabit Ethernet* / *Gigabyte System Network* / *Giganet* / *InfiniBand* /*Myrinet*
Rede de armazenamento (*SAN - Storage Area Network*)	Armazenamento de blocos
	Protocolos
	Sistemas de arquivos
Mainframes x clusters	Aplicações / custo / desempenho / disponibilidade / escalabilidade / manutenabilidade / *software*
Formas de programação de clusters	Paradigma *stream processing*: exploração de paralelismo
	Linguagens: C / CTM / CUDA
	Organização de *threads*: *grid* / *block* / *warp* / *kernel*
	Protocolo MPI (*Message Passing Interface*) Protocolo PVM (*Parallel Virtual Machine*)
Sistemas operacionais para *clusters*	*Linux* versus *Windows*
Paralelismo ILP (*Instruction-Level Parallelism*)	DLP (*Data-Level Parallelism*) e TLP (*Task-Level Parallelism*)

bases remontam ao projeto do referido curso, o primeiro no país a ter o currículo baseado na metodologia PBL. Neste curso, com prazo de integralização em 10 semestres, os estudos integrados, nos quais o método PBL é obrigatoriamente empregado, representam cerca de 30% da carga horária total, de 4345 horas. Em relação aos demais componentes curriculares do curso, as disciplinas das áreas de matemática, física e química são lecionadas de forma tradicional, ao passo que as disciplinas optativas e os projetos anuais (3 no total) apresentam variações de métodos didáticos (inclusive híbridos com PBL), de acordo com as especificidades de cada disciplina.

O fato de um curso, na área de Engenharia, estar baseado em uma metodologia pouco aplicada a cursos nesta área, acarreta conseqüências opostas. Por um lado, dificulta a aquisição de referenciais teóricos mais específicos à área; por outro, potencializa a troca de experiências, através de resultados concretos, obtidos no exercício da metodologia - esta troca de experiências ocorre, em um primeiro momento, dentro do próprio curso.

De forma geral, os professores do curso de Engenharia de Computação da UEFS puderam constatar, a partir dos estudos científicos realizados e das experiências exercidas desde 2003 (ano de implantação do curso), a necessidade de alguns elementos para o bom andamento do curso, conforme descrição seguinte.

O primeiro elemento considerado é o processo de adaptação ao método PBL, o

qual abrange professores e alunos. Todo professor novo no curso passa por uma oficina de uma semana, antes de ministrar aulas ou "tutoriar"um grupo. Nesta oficina, o professor trava contato com o método, através da apresentação e do estudo de suas bases teóricas, abrangência, desafios e formas de avaliação. Além disso, o docente é introduzido à prática do método e à elaboração de problemas, através da análise de resultados de problemas e projetos apresentados nos estudos integrados. Em relação aos alunos, a primeira semana de aulas é reservada exclusivamente para a compreensão e prática do método. Este processo é complementado com o início das atividades nos estudos integrados. Naturalmente, por serem egressos de um sistema educacional tradicional, há certa dificuldade inercial de adaptação, havendo, inclusive, casos de desistência nos primeiros semestres.

O segundo elemento levantado é o comprometimento dos professores-tutores. O papel dos tutores não se limita apenas ao acompanhamento das sessões tutoriais. A necessidade de renovação de problemas, os quais amparam-se em situações do mundo real, exige dos tutores atualização constante de conhecimento e conexão com a realidade esperada para os egressos. Adicionalmente, os tutores de um Estudo Integrado devem ter disponibilidade para reuniões periódicas, as quais são utilizadas para averiguar o andamento do aprendizado dos alunos, assim como as soluções que estão sendo encontradas em grupos tutoriais distintos. Em relação ao foco deste capítulo, os tutores que atuam no Estudo Integrado de Sistemas Digitais necessitam, algumas vezes, projetar e implementar (ao menos parcialmente) os produtos requiridos em problemas que exigem implementação, de forma a visualizar e antecipar eventuais dificuldades para os alunos (a serem clarificadas em aulas de consultoria). Isto ocorreu, por exemplo, em alguns problemas do estudo de casos 1.

O terceiro elemento, necessário ao bom funcionamento da metodologia, é a adequação da complexidade dos problemas. Este é um ponto crítico, causador de muitas queixas pelos alunos - algumas vezes, os problemas são considerados desestimulantes ou simples; outras vezes, são considerados complexos ou super-dimensionados. Uma fonte de problemas simples/desestimulantes é a subestimação da capacidade dos alunos (falta de acompanhamento/conhecimento de problemas já tratados pela turma, incluindo outros estudos integrados). Por outro lado, problemas complexos/superdimensionados surgem por vários motivos, incluindo: prazos de desenvolvimento apertados; especificações extensas e/ou abertas; falhas de formação dos alunos; e presença de distratores. Além disso, sem os cuidados devidos, problemas considerados de média complexidade podem tornar-se simples, mediante uma busca mais detalhada na internet (descoberta de soluções prontas). Boas práticas para a elaboração de um problema incluem: planejamento (separar os conceitos abordados e elaborar o problema com antecedência); realização de buscas na internet (pensar com a cabeça do aluno); e manutenção de um histórico de produtos/relatórios, com observações sobre dificuldades de desenvolvimento em problemas passados.

Em relação aos alunos, além do nível de complexidade dos problemas, uma de suas principais reclamações refere-se à sobrecarga de atividades, uma vez que a resolução dos problemas os mantêm em atividade de estudo constante. Outra queixa refere-se a diferenças de comportamento dos professores-tutores, o que tenta-se amenizar com as reuniões de acompanhamento entre tutores.

Os casos tratados neste capítulo referem-se a uma parcela dos problemas apresentados ao longo desses anos, no Estudo Integrado de Sistemas Digitais. Geralmente, em um semestre, são dados três ou quatro problemas. Devido à extensão do conteúdo programático do referido estudo integrado, não é possível tratar todos os tópicos da ementa nos problemas - por outro lado, também não é possível a total dissociação de conteúdo entre as aulas teóricas e os conceitos trabalhos nos problemas. Assim, o ideal é o planejamento das atividades de ensino antes do início do semestre, para tratar todo o conteúdo programático e, também, evitar possíveis descompassos entre assuntos tratados nos módulos teóricos e nos problemas dos grupos tutoriais. Mesmo com tal planejamento, continua-se dependendo do desenvolvimento da turma: há casos em que os alunos consideram difícil o estudo, por conta própria, de determinados assuntos, julgando que estes deveriam ter sido introduzidos antes, nas aulas teóricas. Em outras situações, nas atividades dos grupos tutoriais, se percebe a não assimilação adequada de alguns assuntos já abordados, semanas antes, nas aulas teóricas.

A assimilação é um ponto importante - percebe-se claramente que os tópicos tratados nos grupos tutoriais são melhor assimilados do que os tratados apenas em aulas teóricas, devido ao tempo que os alunos passam convivendo com um problema (de duas a seis semanas). Disso resulta diferenças entre as turmas, de acordo com os problemas apresentados. Por exemplo, turmas que não receberam um problema curto de linguagem *assembly*, tiveram desempenho inferior no projeto do processador.

Em relação aos assuntos tratados apenas nas aulas teóricas, pode-se destacar superescalaridade e processadores VLIW. Tópicos especiais, alguns ausentes da ementa, foram discutidos em seminários. Conforme visto nos estudo de casos, estes apresentaram resultados variáveis, os quais dependeram, principalmente, da receptividade dos alunos. Neste ponto constatou-se que, ao excluir-se a parte de implementação do produto final, os alunos relaxaram. Esta é uma correção que deverá ser feita, nas próximas oportunidades que seminários forem aplicados.

Conforme pode-se constatar ao longo do capítulo, a metodologia PBL, como qualquer outra de ensino/aprendizagem, possui seus prós e contras. Considerando positiva a experiência que estamos desenvolvendo no curso de Engenharia de Computação da UEFS, podemos afirmar que o uso de PBL influencia de forma ampla a formação de um aluno, não só em termos acadêmicos mas também em seu comportamento individual e em grupo. Os alunos passam a ter um perfil onde se destaca a capacidade de expressar e ouvir opiniões, tomar decisões e comunicar-se de forma colaborativa, entre outras características que podem ser adquiridas mediante a aplicação da metodologia PBL, relativas à sua formação humanística e técnica.

No caso específico da área de Arquitetura de Computadores, o perfil de aluno obtido com a aplicação da metodologia PBL é muito interessante, por tratar-se de uma das áreas cujo escopo e importância são sempre crescentes, o que demanda por profissionais altamente adaptativos, com bons conhecimentos técnicos e desenvolvedores de soluções criativas.

Referências Bibliográficas

Alabanese, M. A., & Mitchell, S. 1993. Problem-Based Learning: A Review of Literature on its Outcomes and Implementation Issues. *Academic Medicine*, **68**, 52–81.

Amador, J.A., Miles, L., & Peters, C. B. 2007. *The Practice of Problem-Based Learning: A Guide to Implementing PBL in the College Classroom*. Anker Publishing Company, Inc.

Angelo, M. F., & Santos, J. A. M. 2009. Análise dos Problemas Aplicados a um Estudo Integrado de Concorrência e Conectividade Utilizando PBL. *In: Anais da VI Conferência Internacional Educação em Engenharia e Computação*.

Bittencourt, R. A., & Figueiredo, O. A. 2003. O Currículo do Curso de Engenharia de Computação da UEFS: Flexibilização e Integração Curricular. *Pages 171–182 of: Anais do XXIII Congresso da Sociedade Brasileira de Computação*, vol. 4. Campinas: SBC.

Bittencourt, R. A., Figueiredo, O. A., & Farias, P. C. M. A. 2002. *Projeto do Curso de Graduação em Engenharia de Computação da UEFS - Universidade Estadual de Feira de Santana*. Tech. rept. UEFS – Universidade Estadual de Feira de Santana.

Bruner, J. S. 1973. *Uma nova teoria da aprendizagem*. Rio de Janeiro: Bloch Editores.

Chang, Ling-Chian, & Lee, Greg C. 2006 (Outubro). Incorporating PBL in a High School Computer Science Course. *Pages 9–14 of: Frontiers in Education Conference*. 36th Annual.

CNE. 2002 (9 de abril). *Resolução CNE/CES 11/2002*. Diário Oficial da União. Seção 1.

Costa, L. R. J., Honkala, M., & Lehtovuori, A. 2007. Applying the Problem-Based Learning Approach to Teach Elementary Circuit Analysis. *IEEE Transactions on Education*, **50**(1).

de Ensino de Engenharia, ABENGE Associação Brasileira. 1991 (maio). *Perfil do Engenheiro do Século XXI*.

Delisle, R. 1997. *How to Use Problem-Based Learning in the Classroom*. Alexandria: Association for Supervision and Curriculum Development.

Dochy, Filip, Segers, Mien, den Bossche, Piet Van, & Gijbels, David. 2003. Effects of problem-based learning: a meta-analysis. *Learning and Instruction*, **13**(October), 533–568.

dos Santos, D. M. B., Pinto, G. R. P. R., Sena, C. P. P., Bertoni, F. C., & Bittencourt, R. A. 2007. Aplicação do método de Aprendizagem Baseada em Problemas no curso de Engenharia da Computação da Universidade Estadual de Feira de Santana. *In: Congresso Brasileiro de Educação em Engenharia - COBENGE*.

Felder, R. M., & Brent, R. 2003. Learning by doing. *Chem. Eng. Educ.*, **37**, 282–283.

Groh, & Duch, Al. 2001. *The Power of Problem-Based Learning*. 1a. edn. FALMER/KP.

Hadgraft, R. 1998. Problem-based learning: A vital step towards a new work Environment. *Int. J. Eng. Educ.*, **14**, 14–23.

Hadgraft, R., & Holecek, D. 1995. Viewpoint: Towards Total Quality Using Problem-Based Learning. *Int. J. Eng. Educ.*, **11**, 8–13.

Hennessy, John L., & Patterson, David A. 2008. *Arquitetura de Computadores: Uma Abordagem Quantitativa*. 4a. edição edn. Editora Campus - Elsevier.

Komatsu, R. S., Zanolli, M. B., Lima, V. V., & Branda, L. A. 1998. *Guia do Processo de Ensino - Aprendizagem: Aprender a Aprender*. 2a. edn. Marília - SP - Brasil: Faculdade de Medicina de Marília.

Mantri, A., Dutt, S., Gupta, J. P., & Chitkara, M. 2008. Design and Evaluation of a PBL-Based Course in Analog Electronics. *IEEE Transactions on Education*, **51**(4), 432–438.

Martinez-Mones, A., Gomez-Sanchez, E., Dimitriadis, Y. A., Jorrin-Abellan, I. M., Rubia-Avi, B., & Vega-Gorgojo, G. 2005. Multiple Case Studies to Enhance Project-Based Learning in a Computer Architecture Course. *IEEE Transactions on Education*, **48**(3), 482–489.

McKeachie, W. J. 1999. *Teaching Tips: Strategies, Research and Theory for College and University Teachers*. Boston, EUA: Houghton-Mifflin Company.

Montero, E., & Gonzalez, M. J. 2009. Student Engagement in a Structured Problem-Based Approach to Learning: A First-Year Electronic Engineering Study Module on Heat Transfer. *IEEE Transactions on Education*, **52**, 214–221.

Northern III, J. 2007 (Abril). Project-Based Learning For a Digital Circuits Design Sequence. *Pages 358–362 of: IEEE Region 5 Technical Conference*. ISBN: 978-1-4244-1280-8.

Oliveira, W. L. A. DE, de Arruda, G. H. M., & Bittencourt, R. A. 2007. Uso do Método PBL no Ensino de Arquitetura de Computadores. *In: Proceedings of the 2007 International Conference on Engineering and Computer Education*.

Pelleh, Moshe, Haberman, Bruria, & Rosenthal, Tammy. 2008. Linking Theory, Practice and System-Level Perception: Using a PBL Approach in an Operating Systems Course. *The Journal of Issues in Informing Science and Information Technology*, **5**, 395–408.

Penaforte, J. 2001. *John Dewey e as Raízes Filosóficas da Aprendizagem Baseada em Problemas*. São Paulo: Hucitec.

Ramos, E.M.F. 1999. *Formação do Engenheiro: Desafios da Atuação Docente, Tendências Curriculares e Questões Contemporâneas da Educação Tecnológica*. Florianópolis: Editora da UFSC.

Ribeiro, L. R. C. 2008a. Aprendizagem baseada em problemas (PBL) na educação em engenharia. *Revista de Ensino de Engenharia*, **27**, 23–32.

Ribeiro, L. R. C. 2008b. *Aprendizagem Baseada em Problemas (PBL): Uma Experiência no Ensino Superior*. São Carlos - SP - Brasil: EdUFSCAR.

Santos, J. A. M., Angelo, M. F., & Loula, A. 2008a. Utilização do método PBL em um Estudo Integrado de Programação. *In: Anais do XXXVI Congresso Brasileiro de Educação em Engenharia*.

Santos, S. C., Batista, M. C. M., Cavalcanti, A. P. C., Albuquerque, J., & Meira, S. R. L. 2008b (Outubro). Usando PBL na Qualificação de Profissionais em Engenharia de Software. *In: Fórum de Educação do XXII Simpósio Brasileiro de Engenharia de Software - SBES*.

Schmidt, H. G. 1993. Foundations of problem-based learning: some explanatory notes. *Medical Education*, **27**(6), 422–432.

Schwartz, Peter. 2001. *Problem Based-Learning: Case Studies, Experience and Practice*. 1a. edn. Routledge.

Solomon, G. 2003. Project-Based learning: A Primer. *Technol. Learn.*, **23**, 20–30.

Striegel, A., & Rover, D. 2002 (novembro). Problem-based Learning in an Introductory Computer-Engineering Course. *Pages F1G7–F1G12 of: Frontiers in Education (FIE)*.

Thomas, J. W., Mergendoller, J. R., & Michaelson, A. 1999. *Project-Based Learning: A Handbook for Middle and High School Teachers*. Buck Institute for Education.

Vernon, D.T.A., & Blake, R. L. 1993. Does Problem-Based Learning Work? A Meta-Analisys of Evaluative Research. *Academic Medicine*, **68**, 550–563.

Woods, D. R. 1995. *Problem-Based Learning: Helping Your Students Gain the Most From PBL*. Hamilton, ON, Canada: Waterdown.

Capítulo 3

Ensino de arquiteturas de processadores many-core e memórias cache utilizando o simulador Simics

Marco Antonio Zanata Alves (UFRGS - mazalves@inf.ufrgs.br)
Henrique Cota de Freitas (PUC Minas - cota@pucminas.br)
Philippe Olivier Alexandre Navaux (UFRGS - navaux@inf.ufrgs.br)

Abstract

Concerning the rise of new generations of many-core processors with the possibility of tens or hundreds of processor cores for general purpose, the teaching of processor architecture becomes a great challenge, especially considering the rapid development of the area. In this context, there is also several design alternatives with new concepts and techniques that increase the exploration space for students in disciplines related to the computer architectures. This chapter presents an approach of using a full system simulation environment, called Simics in the classroom, focusing on design, simulation and evaluation of architectures of many-core processors. To support this approach, we used our experience of evaluating of shared cache memory on many-core processors, which will be presented throughout this chapter, together with results obtained by students in the classroom.

Resumo

Com o surgimento das novas gerações de processadores many-core *com possibilidade de dezenas ou centenas de núcleos de processamento de propósito geral, o ensino de arquitetura de processadores se torna um grande desafio, principalmente considerando a rápida evolução da área. Neste contexto, surgem também várias alternativas de projeto com novos conceitos e técnicas que aumentam o espaço de exploração dos alunos nas disciplinas relacionadas à arquitetura de computadores. Este capítulo apresenta uma abordagem de uso de um ambiente de simulação de sistema completo chamado Simics em sala de aula, focando no projeto, simulação e avaliação das arquiteturas de processadores* many-core*. Para dar suporte a esta abordagem, foram utilizadas as experiências de avaliação de memórias* cache *compartilhadas em processadores* many-core*, a qual será apresentada ao longo deste capítulo, juntamente com resultados obtidos por alunos em sala de aula.*

3.1. Introdução

A busca por desempenho computacional tem aumentado ao longo das décadas, através da exploração de técnicas como *pipeline*, superescalaridade e *multithreading*. Estas técnicas aumentam o paralelismo de execução das aplicações e assim reduzem o tempo de execução e obtenção de resultados. Com base nestas técnicas, o projeto de uma arquitetura pode focar o paralelismo no nível de instrução ou *threads*. Arquiteturas superescalares têm como característica principal a execução paralela de instruções e exploram o aumento da freqüência de operação para atingir altas taxas de desempenho. O suporte a múltiplas *threads* [1] [2] é uma alternativa de exploração de paralelismo não mais no nível de instruções, mas no nível de fluxo de instruções (*threads*). A principal diferença entre arquiteturas superescalares e *multithreading* pode ser explicada pelas cargas de trabalho. Cargas com alto paralelismo no nível de instruções podem ser melhores suportadas por arquiteturas superescalares, enquanto cargas com alto paralelismo no nível de *threads* são melhores suportadas por arquiteturas *multithreading*.

As abordagens *single-threading* e superescalares tradicionais vêm dando lugar a abordagens diferentes a fim de aumentar ainda mais o desempenho e reduzir o consumo de potência. O projeto de processadores com técnicas *multithreading* e múltiplos núcleos de processamento mais simples (arquiteturas com pouco suporte de paralelismo no nível de instrução) vem sendo consolidado como uma alternativa para o aumento do desempenho computacional. Para a nova geração de processadores com arquiteturas *many-core*, a quantidade elevada de núcleos demanda um novo tipo de projeto baseado em um sistema heterogêneo com diversos núcleos diferentes, memórias *cache*, redes de interconexões, protocolos de coerência adequados, a fim de obter um alto desempenho no sistema final.

Neste contexto de arquiteturas *many-core*, fica clara a necessidade do ensino de diversos novos conceitos aos alunos nas disciplinas de arquiteturas de computadores. Entretanto, essa tarefa de ensino pode ser um pouco abstrata, caso os alunos não consigam aplicar e trabalhar, mesmo que em ambiente simulado, os conceitos aprendidos em sala de aula.

Portanto, o objetivo principal desse capítulo é apresentar uma experiência de ensino e avaliação de processadores *many-core* através de um ambiente de simulação que propicie aos alunos e professores a exploração de diferentes configurações de arquiteturas e organização. Para isso, será apresentado de forma descritiva como utilizar um ambiente de simulação completo em sala de aula, além das possíveis métricas de avaliação de desempenho que podem ser aplicadas. A meta final é difundir as experiências relacionadas ao uso de um simulador completo em sala de aula tendo como foco uma melhor formação dos alunos para próxima geração de processadores *many-core*. Para isso, os métodos apresentados foram definidos a partir do trabalho de pesquisa que visava a avaliação de memórias *cache* compartilhadas em diferentes arquiteturas de processadores *many-core*, e dos resultados de avaliações dos trabalhos executados pelos alunos.

Este capítulo é organizado da seguinte forma. A Seção 3.2 apresenta uma breve introdução sobre arquiteturas de processadores *multi-core* e *many-core*. A Seção 3.3 trará conceitos relacionados as arquiteturas de memória *cache* compartilhadas, que será o objeto de estudo desse capítulo. A Seção 3.4 irá apresentar um método para avaliação de memória *cache* compartilhadas. A Seção 3.5 mostra a importância de ambientes de simu-

lação de sistemas completos. A Seção 3.6 foca principalmente na apresentação do Simics. A Seção 3.7 apresenta alguns estudos de caso. Por fim, a Seção 3.8 trará as principais conclusões relacionadas às experiências obtidas.

3.2. Arquiteturas de Processadores *Multi-Core* e *Many-Core*

O estudo das atuais arquiteturas de processadores *multi-core* disponíveis no mercado é de grande importância para avaliar a direção traçada atualmente pelas indústrias de processadores. Um dos pontos principais para o desempenho do processador nas arquiteturas *multi-core* está na organização da memória *cache*. As pesquisas em desenvolvimento e os processadores comerciais apontam diferenças na influência do compartilhamento e dos níveis de memórias *cache* escolhidas. Nesta seção são abordadas as principais características dos projetos de processadores com uma visão evolutiva e dos problemas relacionados na escolha de uma característica específica.

Vários projetos de arquiteturas de processadores vêm adotando ao longo de décadas diversas técnicas tradicionais [3] [4] [5] [2] [1] como *pipeline*, superescalaridade e *multithreading* para explorar o paralelismo de execução das aplicações e assim melhorar o desempenho de computadores de propósito geral.

Em um *pipeline* superescalar [4], além do processamento de instruções ser dividido em estágios, é feita uma completa sobreposição das instruções, utilizando para isso, o aumento do número de unidades funcionais e técnicas para solucionar falsas dependências entre as instruções, dentre outras. Desta forma, os processadores superescalares são capazes de aumentar consideravelmente o desempenho na execução de cargas de trabalho com alto paralelismo no nível de instruções.

O suporte a múltiplas *threads* [2] [1] é uma alternativa de exploração de paralelismo não mais no nível de instruções, mas no nível de fluxo de instruções (*threads*). Isso significa um aumento na vazão de *threads*, podendo mais de uma *thread* ser executada ao mesmo tempo, ao contrário da superescalaridade onde a vazão é de instruções de uma única *thread*. Diversas são as técnicas para exploração do paralelismo no nível de *threads*, sendo que a mais conhecida é a SMT (*Simultaneous Multithreading*) que é suportada por uma arquitetura superescalar.

Ao longo de décadas, as técnicas de aumento de profundidade do *pipeline* aliado ao aumento da freqüência de trabalho do processador (ciclo de relógio) foram utilizadas a fim de obter o máximo desempenho a cada nova geração de processador. O custo para esse ganho de desempenho foi o aumento da complexidade da unidade de controle dos processadores, assim como o aumento no consumo de potência e temperatura do sistema computacional. Entretanto, esta forma tradicional de aumento de desempenho começou a apresentar problemas físicos, relacionados com o alto grau de integração dos componentes como o atraso do fio, consumo de potência estática, entre outros. Assim, essa complexa abordagem de extração de paralelismo vem dando lugar a uma abordagem diferente. A fim de aumentar ainda mais o desempenho, e ainda, algumas vezes diminuir a potência dissipada, o uso de processadores com múltiplos núcleos (CMPs - *Chip Multi-Processors*) [6] vem sendo consolidada [7] [8] [9] [10] como uma boa abordagem para o aumento do desempenho de computação.

Ainda com a técnica de múltiplos núcleos no mesmo *chip*, nada impede que sejam utilizadas superescalaridade e SMT em cada núcleo, mas nesses casos pode haver um aumento considerável na área do CMP em função da duplicação de registradores além de outros componentes. Em função desta nova abordagem de projeto, os processadores com múltiplos núcleos e que suportam múltiplas *threads* também são conhecidos como CMT ou *Chip Multithreading* [11] [12].

3.2.1. Processadores *Multithreading*

Nos projetos atuais de processadores, um dos grandes objetivos é a extração máxima do desempenho. Uma das formas está na exploração do paralelismo, seja na execução das instruções ou nos fluxos de instruções. Nesse contexto, podemos considerar que um fluxo de instruções é uma *thread* e que uma *thread* é um processo, ou parte de um programa em execução. Se um processador suporta a execução de múltiplas *threads* [13] [14] [15] [16], significa que esse processador é capaz de executar fluxos de instruções diferentes. Nesse caso, cada uma destas *threads*, ou fluxo de instruções, inicia em endereços diferentes de memória.

O suporte à *multithreading* possui duas abordagens [1] [2]: *Implicit Multithreading* e *Explicit Multithreading*:

- *Implicit Multithreading*: Exploração do paralelismo existente em programas seqüenciais através de especulação no nível de *thread*. Nessa abordagem, um processador gera múltiplas *threads* especulativas de um único programa seqüencial, dinamicamente, ou estaticamente com ajuda do compilador, e executa todas concorrentemente.
- *Explicit Multithreading*: Exploração do paralelismo existente entre programas de origens diferentes. As *threads* geradas a partir de cada um desses programas podem ser executadas em um mesmo *pipeline*.

Nos dois casos, cada uma das *threads* possui um banco de registradores e contadores de programa específicos, representando cada um dos múltiplos contextos em atividade no processador. A diferença está no uso de execução especulativa de *threads* de um mesmo programa seqüencial na abordagem *implicit multithreading* ou na execução de *threads* independentes e de programas distintos na abordagem *explicit multithreading*.

Um único núcleo de processamento (escalar ou superescalar) pode suportar múltiplas *threads*. Pequenas modificações na organização interna do núcleo são responsáveis por permitir a execução simultânea ou chaveada das *threads*.

Ainda em se tratando de *multithreading*, podemos ter diversos tipos de exploração de paralelismo no nível de *threads*, como SMT (*Simultaneous Multithreading*), IMT (*Interleaved Multithreading*) e BMT (*Block Multithreading*).

- *Simultaneous Multithreading*: Esse tipo avançado de *multithreading* se aplica a processadores superescalares. Um processador superescalar simples busca instruções da mesma *thread* a cada ciclo do processador. Em um processador SMT, o processador pode buscar instruções de várias *threads* a cada ciclo do processador. Esse

tipo de *multithreading* se prevalece do fato de que, para uma única *thread*, o número de instruções paralelas a serem buscadas e executadas é limitado. Assim, buscando instruções de múltiplas *threads*, o processador tenta reduzir o número de unidades funcionais sem uso a cada ciclo de relógio.

- *Interleaved Multithreading*: A execução de cada *thread* é alternada em cada ciclo de relógio do processador. Esse modo de trabalho visa remover todas paradas por dependência de dados de um *pipeline*. Uma vez que uma *thread* é relativamente independente das outras *threads*, existe menos chance de uma instrução precisar esperar por um dado de uma instrução executada anteriormente, uma vez que existirá instruções de outras *threads* intercaladas no *pipeline*.

- *Block Multithreading*: Cada *thread* é executada até que seja bloqueada por um evento que normalmente cria um longo período de espera. Tal evento pode ser uma falta de dados na memória *cache* criando assim uma latência para acesso aos dados. Dessa maneira, ao invés de esperar o evento de alta latência, o processador deverá trocar a execução para outra *thread* que esteja pronta para execução. A *thread* que ocasionou o evento de alta latência só receberá status de pronta para execução assim que sair do estado de espera. Esse modelo de *multithreading* visa esconder as altas latências de acesso à memória, mascarando essas latências com a execução de outro fluxo de instruções.

3.2.2. Processadores *Multi-Core*

Pesquisas sobre as melhores alternativas de projeto de arquiteturas de processadores têm usado basicamente o estudo de cargas de trabalho para entender melhor o comportamento do processador.

Um dos primeiros estudos que identificou o potencial do uso de *chip multiprocessor* foi proposto em [6], onde foi apresentado um estudo onde dois tipos de arquiteturas foram expostos a um mesmo tipo de carga de trabalho. O estudo procurou definir qual o melhor tipo de arquitetura para cargas onde havia um baixo ou grande paralelismo no nível de *thread*. A Figura 3.1 [6] apresenta os dois tipos de arquiteturas que foram comparadas, uma arquitetura superescalar com execução de seis instruções simultâneas e uma arquitetura com múltiplos núcleos de processamento suportando duas instruções simultâneas por núcleo de processamento.

Para garantir apenas a influência da carga de trabalho submetida, os dois projetos possuíam as mesmas latências de acessos à memória principal, mesmo tempo de acesso à memória *cache* e a mesma ocupação de área. Então, as mesmas cargas de trabalho com características de operações de números inteiros, ponto flutuante, e de multiprogramação foram aplicadas aos dois projetos.

Os resultados demonstraram que para cargas de trabalho onde os fluxos de instruções das aplicações não são paralelizáveis, o ganho é favorável ao processador superescalar em 30%. Nesse caso existe uma exploração melhor do paralelismo no nível de instrução. Para aplicações onde existe um baixo paralelismo de *threads*, o ganho ainda é favorável à arquitetura superescalar, mas no máximo de 10%. No entanto, onde há um

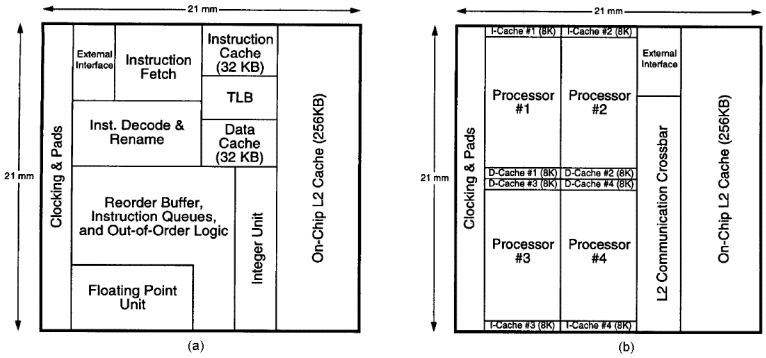


Figura 3.1. Comparativo de processadores [6], (a) Processador superescalar, (b) Processador *multi-core*.

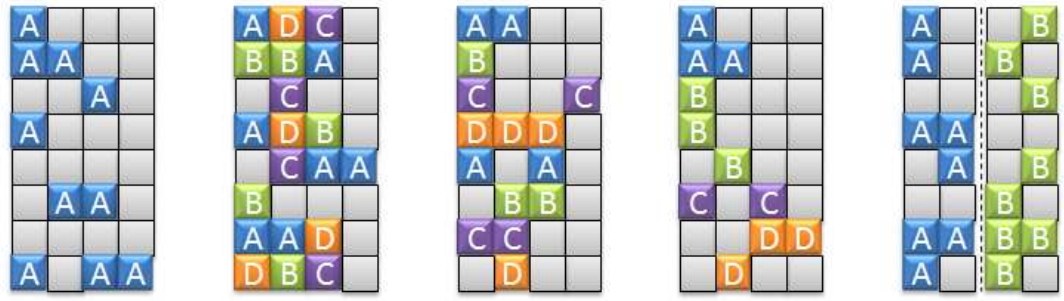
grande paralelismo no nível de *thread*, o ganho passa a ser da arquitetura CMP variando de 50% a 100% em relação ao superescalar.

As cargas de trabalho com grandes níveis de paralelismo em *thread* executam aplicações independentes com processos independentes. Aplicações de visualização e multimídia, processamento de transações e aplicações científicas de ponto flutuante são exemplos destas cargas de trabalho. Esta pesquisa serviu como base para o processador Hydra CMP [17], e também gerou resultados para o processador UltraSparc-T1 [9].

Atualmente, a grande maioria dos processadores de propósito geral são exemplos de arquiteturas com núcleos homogêneos (iguais) e para um mesmo propósito de funcionamento (aplicações gerais no caso do GPP - *General-Purpose Processor*). No entanto, projetos de processadores *multi-core* para aplicações em sistemas embarcados, freqüentemente possuem núcleos heterogêneos [18] [10]. Nesse caso, cada núcleo, ou conjunto de núcleos, é responsável por processamentos específicos e distintos dos demais. Uma classe de processadores que representa adequadamente as arquiteturas com núcleos heterogêneos são os processadores conhecidos como MPSoCs (*Multi-Processor System-on-Chip*) [19]. Estes processadores podem apresentar mais de um processador GPP (*General-Purpose Processor*) ou ASIP (*Application Specific Instruction Set Architecture*) interno ao *chip*, porém cada um desses possui características diferentes, onde um determinado núcleo pode se adequar melhor a um conjunto de aplicações enquanto outro núcleo se encaixa melhor a um segundo conjunto de aplicações.

A Figura 3.2, estendida de [2] apresenta uma comparação entre um superescalar de quatro vias 3.2(a), um processador SMT (*Simultaneous Multithreading*) 3.2(b), um processador IMT (*Interleaved Multithreading*) 3.2(c), um processador BMT (*Block Multithreading*) 3.2(d), sendo todos CMT superescalares de quatro vias, e por fim, um processador *multi-core* onde cada um dos dois núcleos é um superescalar de duas vias 3.2(e).

Fora o processador superescalar simples, todos os processadores são CMTs que suportam de duas (CMP) até quatro *threads* (SMT, IMT e BMT) simultâneas. No processador superescalar simples apenas uma *thread* está ativa por vez (A), assim apenas instruções de uma *thread* são executadas até que haja uma troca de contexto. Para o processador SMT, quatro *threads* ficam ativas (A, B, C e D), além disso, instruções das



(a) Superescalar (b) SMT (c) IMT (d) BMT (e) CMP

Figura 3.2. Comparativo entre processadores superescalar, processadores *multithreading* e *multi-core*. Na vertical está ilustrado a linha do tempo de execução, na horizontal encontram-se as vias de execução de cada processador. Pode-se ver ainda, as vias de execução ocupadas com os fluxos de instrução (A, B, C e D).

quatro *threads* podem ser executadas em um mesmo ciclo de máquina aproveitando todas as unidades funcionais. Já no caso do IMT e BMT, apenas instruções de uma *thread* são executadas por ciclo, onde a troca entre *threads* ativas acontece a cada ciclo no caso do IMT e para o BMT a troca ocorre ao executar um evento de alta latência. No caso do *multi-core* ou CMP (*Chip Multiprocessor*) cada núcleo recebe uma única *thread*, mas duas instruções de cada *thread* podem ser executadas ao mesmo tempo. Assim, para que um núcleo passe a operar sobre outra *thread*, deve haver uma troca de contexto, o que é um evento de mais alta latência que a troca entre *threads* ativas.

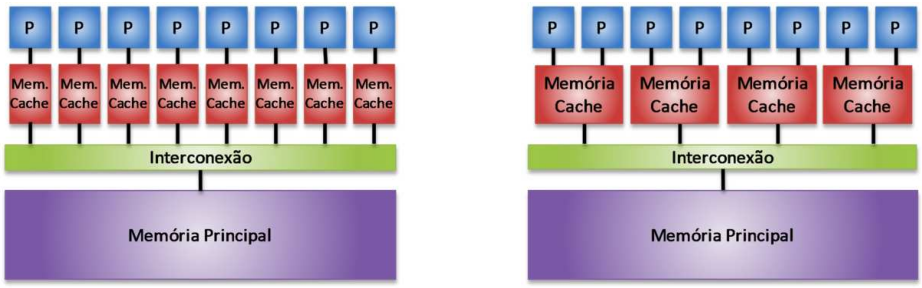
3.3. Arquiteturas de Memórias *Cache* Compartilhadas

Devido ao aumento do número de núcleos de processamento dentro do processador, novos modos de organização da hierarquia de memória estão sendo estudados. As memórias *cache* para esses processadores podem ser privadas ou compartilhadas conforme apresenta a Figura 3.3.

Em um modelo de organização de memória *cache* privada cada processador possui sua própria hierarquia de memória *cache* totalmente isolada, como apresentado na Figura 3.3(a), somente interconectada aos demais núcleos por meio da memória principal.

No modelo de memória *cache* compartilhada, ilustrado na Figura 3.3(b), existe pelo menos um nível na hierarquia de memórias *cache* compartilhada entre dois ou mais núcleos de processamento. No caso ilustrado, a memória *cache* está compartilhada a cada dois núcleos de processamento.

A utilização de modelos de memórias *cache* privadas ou compartilhadas influencia no projeto do sistema, uma vez que questões sobre a implementação do protocolo de coerência e consistência entre as memórias, além da quantidade de portas de entrada e saída necessárias na memória *cache*, largura de banda de interconexão e área do projeto, são afetados pelo modelo de organização de memória privada ou compartilhada.



(a) Memórias *cache* privadas. (b) Memórias *cache* compartilhada.

Figura 3.3. Diagrama representando organizações de memórias *cache*, apresentando memórias *cache* privadas 3.3(a) e compartilhadas 3.3(b).

O desempenho do sistema também pode ser influenciado pelo modo de compartilhamento de memória *cache* adotado, uma vez que o compartilhamento poderá favorecer o desempenho reduzindo o número de faltas quando os processadores que compartilham estiverem trabalhando no mesmo conjunto de dados. Por outro lado, a quantidade de faltas de dados relacionadas ao conflito de endereços e capacidade pode aumentar se os dados em uso não forem os mesmos nos processadores que compartilham a memória *cache*.

Juntamente com esse modelo de memória compartilhada entre vários núcleos, algumas preocupações foram inseridas durante o projeto de uma memória *cache* para os novos processadores com múltiplos núcleos de processamento. Algumas das questões que ganharam mais força nesse novo modelo de memória foram o controle de coerência entre memórias *cache*, estrutura de interconexão, portas de entrada e saída para as memórias, entre outras.

3.3.1. Coerência e Consistência da Memória *Cache*

Ao tratar de coerência de dados em memórias *cache*, estamos envolvendo dois aspectos importantes. O primeiro diz respeito à coerência, que define quais valores devem ser retornados durante a leitura. O segundo aspecto é a consistência, que determina quando um valor gravado será retornado por uma leitura.

Assim, de acordo com [3], um sistema de memória é considerado consistente se:

- A ordem do programa for preservada. Uma leitura pelo processador P_i em uma posição X após uma gravação por P_j em X, sem a ocorrência de gravações em X por outro processador entre a gravação e leitura por P_i, sempre retorna o valor gravado por P_j.

- Existe uma visão coerente da memória. Uma dada leitura da posição X pelo processador P_i após uma gravação em X por P_j retorna o valor gravado por P_j se a leitura e a gravação estiverem bastante separadas no tempo e não ocorrer nenhuma outra gravação em X entre os dois acessos.

- Gravações na mesma posição são serializadas. Duas gravações na mesma posição

por dois processadores quaisquer serão vistas na mesma ordem por todos os processadores.

Mesmo que o modelo ideal seja que, após um processador gravar um dado na memória, esse seja atualizado instantaneamente em todas as suas cópias, esse modelo talvez seja impossível de ser implementado. A noção de quando um dado gravado será visível para leituras é tratada pela consistência da memória.

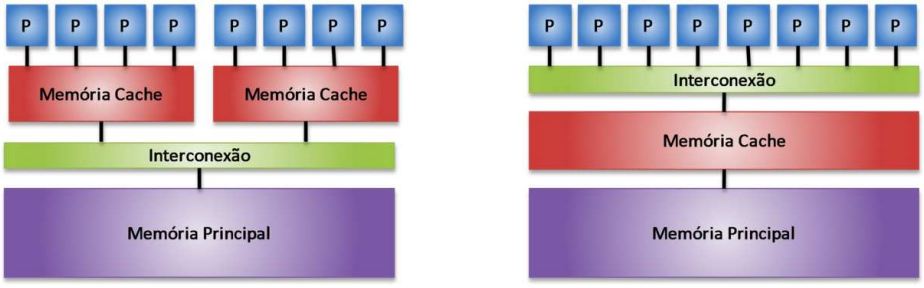
Assim, podemos notar que a noção de coerência e consistência são complementares, pois enquanto a coerência define o comportamento de leituras e gravações em uma mesma posição, a consistência define o comportamento de leituras e gravações em relação a acessos a outras posições de memória.

3.3.2. Protocolos de Coerência

Em um multiprocessador coerente, as memórias *cache* fornecem tanto a migração quanto a replicação de dados compartilhados. Dessa forma, mesmo que existam diversos processadores trabalhando sobre os mesmos dados, cada processador deverá agir sem se preocupar com a existência de outras cópias dos mesmos dados em outros processadores.

As memórias *cache* coerentes proporcionam migração, pois qualquer dado pode ser movido para a *cache* e operado de forma transparente. Essas memórias *cache* também proporcionam replicação, pois diversas memórias *cache* podem conter cópias de um dado referentes ao mesmo endereço.

Dessa forma, é importante ressaltar que, de acordo com [20], o problema de coerência de dados apresenta-se apenas em arquiteturas multiprocessadas que associam memórias *cache* a cada um dos processadores do sistema. Assim, em arquiteturas onde a memória *cache* está associada somente à memória principal, não existem problemas de coerência de memória *cache*. Podemos ver um diagrama da arquitetura com memórias *cache* associadas em blocos ou associada somente à memória principal na Figura 3.4, onde se pode observar duas arquiteturas de memória *cache*, uma com problema de coerência de dados 3.4(a) e outra sem problemas de coerência de dados 3.4(b) [20].



(a) Com problemas de coerência. (b) Sem problemas de coerência.

Figura 3.4. Arquiteturas de memória *cache* com problema de coerência de dados 3.4(a) e sem problemas de coerência de dados 3.4(b), adaptado de [20].

Para que uma memória *cache* seja capaz de manter a coerência de dados, é necessária a adoção de um protocolo de coerência para assegurar o controle de compartilhamento

de qualquer bloco de dados entre várias memórias *cache* e processadores. Existem duas classes principais de protocolos atualmente em uso para assegurar a coerência de memórias *cache*:

- Baseado em Diretório – O status sobre o compartilhamento de um bloco de memória é alocado apenas em uma posição chamada diretório.

- *Snooping* – O status sobre o compartilhamento de um bloco é replicado em todas as memórias *cache* que possuem a cópia de cada dado. Assim, não existe nenhum diretório centralizador. Geralmente todas as memórias *cache* encontram-se em um mesmo barramento e todos espionam (*snoop*) a movimentação no barramento, no caso de alguma outra memória possuir cópia de blocos solicitados.

Como em diversos projetos de multiprocessadores as memórias *cache* estão conectadas a uma única memória principal interconectadas por um barramento, os protocolos de coerência de memória *cache* mais populares são os baseados na técnica de *snooping*, pois tiram proveito da infra-estrutura pré-existente. Como exemplo desse tipo de protocolo pode-se citar o protocolo MESI [5], sendo que o nome MESI vêm das iniciais dos estados possíveis de um dado durante operação (*modified, exclusive, shared*, e *invalid*).

Os pontos-chave [21] de um barramento que possui suporte à coerência são que todas as transações devem ser visíveis para todos os controladores de memória *cache*. Assim, cabe ao protocolo de coerência garantir que todas as transações de memória apareçam no barramento e que todos os controladores façam as ações necessárias ao verem transações relevantes.

3.3.3. Modelos de Consistência

Os modelos de consistência de memória *cache* tratam o grau de consistência que deverá existir no sistema, apresentando soluções sobre quando um valor atualizado em uma memória *cache* qualquer deverá ser visível aos outros processadores.

Embora a questão sobre a consistência de dados na memória *cache* pareça trivial, é de um elevado grau de complexidade, pois tratar a consistência de dados significa escolher o momento em que as variáveis devam ser atualizadas nas suas cópias. Porém, essa escolha envolve também o algoritmo executado nos diversos processadores, além de ser um tratamento sobre condição de corrida, uma vez que pode haver disputa por recursos.

O modelo mais simples de consistência de dados é o chamado consistência seqüencial, que é implementado de forma simples, e exige que um processador retarde a conclusão de qualquer acesso à memória até que todas as invalidações causadas se completem.

Outro tipo de modelo também difundido é o modelo de consistência relaxado [22], que envolve o programador ao gerar programas paralelos e consiste em permitir que as leituras e gravações se completem fora de ordem, mas requer o uso de operações de sincronização para impor ordenação, fazendo com que o programa seja sincronizado e comporte-se como se estivesse em um processador mono-processado.

3.3.4. Interconexões

Como os diversos dispositivos de um sistema, processadores, memórias, portas de entrada e saída e demais dispositivos, precisam se comunicar, essa é a tarefa das interconexões. Algumas das principais características de comparação entre interconexões são [20]:

- Escalabilidade – É uma característica desejável em toda interconexão. Diz respeito a adaptabilidade da interconexão ao aumento da quantidade de dispositivos e também de carga total de trabalho. Assim, um sistema escalável será de fácil expansão mantendo as características principais inalteradas.
- Desempenho – O desempenho desejável de uma interconexão é que essa consiga manipular e dar vazão a todos os dados em tempo hábil. Assim, o desempenho indica a capacidade e a velocidade da transferência de dados pela interconexão.
- Custo – O custo de uma interconexão costuma variar proporcionalmente em função do número de dispositivos interconectados e a capacidade de vazão e latência. Em alguns casos, o custo de uma interconexão pode se referir à área ocupada pelo projeto ou pela potência consumida.
- Confiabilidade – Pode ser tratada como a probabilidade da interconexão atender, de forma adequada, aos dispositivos comunicantes. Assim, a existência de caminhos alternativos ou redundantes entre dispositivos aumenta a confiabilidade da interconexão.
- Funcionalidade – Diz respeito às funcionalidades específicas da interconexão agregadas à transferência de dados. Assim, uma interconexão pode implementar outros serviços como *buffers* de entrada e/ou saída, garantia de ordenação na transferência, ou até mesmo roteamento automático.
- Reusabilidade – Trata da capacidade da interconexão se conectar a diferentes tipos de dispositivos, e também da capacidade de que a cada nova geração o projeto poder ser em grande parte reutilizado.

Uma interconexão pode ser feita de diversas formas e mesmo assim, durante anos, a solução adotada tem sido a interconexão por barramento. Porém, atualmente existem diversas formas de interconexões que estão sendo estudadas a fim de serem implementadas nos processadores multiprocessados, que é o caso das interconexões por matrizes de chaveamento, ou por redes *intra-chip*, também conhecidas como NoC (*Network-on-Chip*) [23].

3.3.5. Barramentos

Um barramento é um canal de comunicação compartilhado que utiliza um conjunto de fios para conectar diversos dispositivos ou subsistemas. As principais vantagens dessa interconexão é a simplicidade, reusabilidade e o baixo custo.

A simplicidade do barramento é considerada a partir do ponto de vista de que novos dispositivos podem ser agregados facilmente sem grandes modificações, porém, o

barramento não é totalmente escalável, pois o mesmo não se adapta para aumentar a vazão de dados ao adicionar um novo componente interconectado.

Quanto à reusabilidade, como o projeto de barramentos muitas vezes segue a determinados padrões, esses podem ser reusados a cada nova geração, com pouca ou nenhuma modificação. Como o projeto de um barramento pode ser visto de forma simplista como uma forma de compartilhamento de um conjunto de fios, essa interconexão é de baixo custo.

A forma clássica de um barramento apresenta dois conjuntos de linhas de comunicação, um conjunto de controle e outro conjunto de linhas de dados como é ilustrado na Figura 3.5. As linhas de controle são usadas para gerenciar as requisições e confirmações, além de informar o tipo dos dados que estão trafegando nas linhas de dados. As linhas de dados são apenas linhas de propagação de informações entre origem e destino.

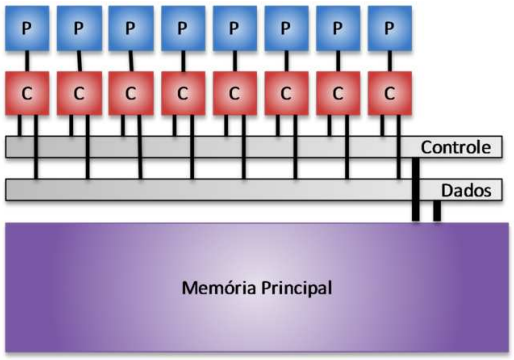


Figura 3.5. Diagrama de um barramento com canais de controle e dados interligando processadores e memória *cache* a memória principal.

Como diversos componentes estão compartilhando um mesmo meio físico do barramento, o protocolo do barramento é contido nas linhas de controle, as quais implementam qual será a política de uso do barramento. O barramento também pode ser classificado como rede dinâmica de interconexão, uma vez que a topologia de comunicação não existe *a priori*.

3.3.6. Matrizes de Chaveamento

A matriz de chaveamento, também conhecida como *crossbar* ou *crossbar switch* possui um custo elevado, porém, também é simples. Assim, embora uma rede de interconexão possa ocupar grandes áreas do projeto para interligar um grande número de dispositivos, essa interconexão se mantém simples uma vez que ao adicionar novas portas para dispositivos não adiciona muita complexidade no controle. No entanto, a chave *crossbar* assim como o barramento não é escalável como solução de interconexão global, o que dificulta um projeto com quantidade elevada de núcleos de processamento.

A Figura 3.6 apresenta uma ilustração de uso de uma chave *crossbar* interconectando memórias *cache* de primeiro nível de diversos processadores com memórias *cache* L2 compartilhadas, cada um possuindo duas portas de comunicação. Dessa maneira, apenas dois núcleos podem ser conectados a cada banco de memória *cache* simultaneamente.

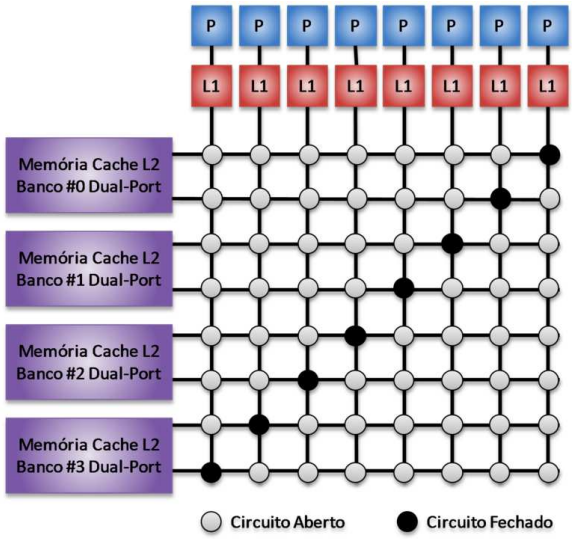


Figura 3.6. Diagrama de uma matriz de chaveamento interconectando processadores e memórias *cache*.

Em uma rede de chaveamento pode se interconectar dois dispositivos quaisquer, desde que esses não se encontrem já ocupados. Uma alternativa para o alto custo é a utilização de várias matrizes de chaveamento em formação hierárquica.

3.3.7. Redes *Intra-Chip*

Com a promessa de rápido aumento no número de núcleos de processamento dentro do processador, aumenta a motivação de estudos sobre formas diferentes de interconexão entre os vários dispositivos dentro do *chip*.

Com o intuito principal de aumentar a escalabilidade e reduzir o custo da interconexão, diversos estudos abordam as redes de interconexão *intra-chip*. Em muitos casos, essa rede de interconexão é formada por diversos roteadores, um em cada dispositivo, interligados. Logo, podem não existir ligações diretas entre todos os dispositivos, sendo necessário então que um pacote trafegue entre os roteadores para chegar ao seu destino.

A política de roteamento da interconexão é a que determina como os pacotes serão chaveados para chegar ao destino. Assim, a forma de chaveamento depende da topologia da rede de interconexão.

A Figura 3.7 apresenta um diagrama de utilização de uma rede de interconexão *intra-chip* de tamanho 5x4 (20 roteadores), conectando diversos núcleos de processamento a quatro bancos de memória *cache*. No caso ilustrado, o tempo de comunicação entre núcleo e memória *cache* pode variar dependendo da localização do núcleo na rede de interconexão, onde os núcleos mais próximos irão conseguir acessar a memória *cache* com apenas dois saltos (*hops*), ou seja, passando apenas pelo roteador local e roteador da memória *cache*, enquanto que nos piores casos, a comunicação poderá custar de 5 até 8 saltos, dependendo do banco de memória que o núcleo precisar acessar.

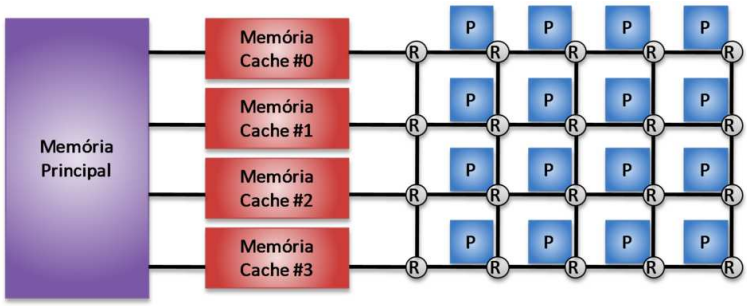


Figura 3.7. Diagrama de uma rede de interconexão *intra-chip* (NoC) interligando diversos núcleos de processamento.

3.4. Método de Avaliação de Memórias *Cache* Compartilhadas

De acordo com [24], a utilização de uma correta metodologia para avaliação de desempenho de sistemas computacionais evita diversos problemas, como a falta de objetivos, objetivos viciados, abordagem não sistemática, análise sem compreender o problema, métricas incorretas de desempenho, carga de trabalho não representativa, técnicas erradas para avaliação, descaso com parâmetros importantes, ignorar fatores significantes, projeto experimental inapropriado, entre outros. São diversos problemas que podem ocorrer, caso não haja um correto planejamento dos experimentos.

Para evitar problemas em nosso projeto de avaliação, podemos seguir diversos passos para assegurar que estamos efetuando as análises com rigor acadêmico. As etapas a seguir foram adaptadas de [24] e aplicadas ao nosso projeto.

3.4.1. Definição do Sistema e Serviços

Esta etapa visa definir os objetivos do estudo delimitando o sistema a ser avaliado. A Figura 3.8 apresenta um diagrama genérico do sistema de memória *cache* a ser avaliado, onde estão definidos por blocos os principais componentes do sistema, como os núcleos de processamento, a interconexão, o sistema de memória *cache* e a memória principal, delimitando ainda o escopo que compreende ao processador, sendo que tais componentes estão em um mesmo *chip*.

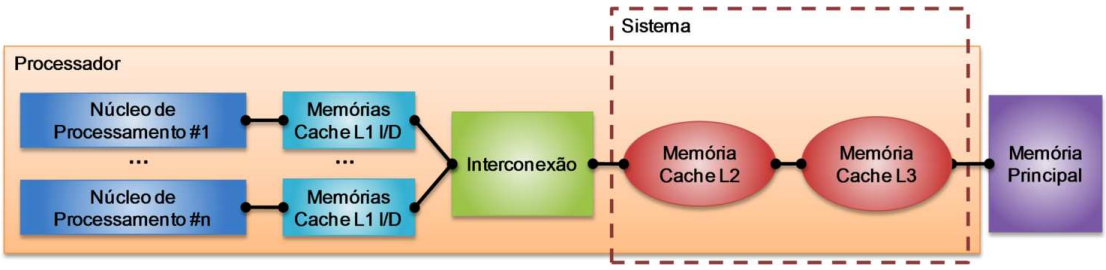


Figura 3.8. Definição do sistema a ser estudado.

Mesmo sem a definição clara da arquitetura e organização do sistema, com a visualização do diagrama do sistema a ser avaliado ficam evidentes os componentes não

principais, mas que são relevantes ao estudo (núcleos de processamento, interconexão e memória principal), e o componente principal do estudo que é o próprio sistema de memória *cache*. Assim, o objetivo final do estudo é indicar formas de reduzir o impacto do sistema de memória *cache* e melhorar o desempenho final do sistema.

Os serviços disponíveis aos núcleos de processamento através da interconexão ao sistema de memória *cache* são definidos de forma sucinta como leitura e escrita de dados. Estes serviços podem ser propagados até a memória principal. Os serviços devem oferecer alto desempenho, evitando assim eventuais paradas dos núcleos de processamento.

3.4.2. Métricas de Avaliação

A escolha das métricas do sistema é importante, uma vez que através dessas métricas as análises serão feitas e assim deve ser possível indicar qual sistema é apropriado para cada situação, ou seja, as métricas devem ser pontos de comparação entre os sistemas avaliados.

Algumas métricas para o sistema de memória *cache* são:

- Métricas de Desempenho
  - Faltas de leituras de dados na memória *cache*.
  - Faltas de escritas de dados na memória *cache*.
  - Porcentagem de faltas de dados na memória *cache*.
  - Faltas de dados na memória *cache* a cada mil instruções executadas (MPKI).
  - Total de ciclos para a execução da carga de trabalho.
  - Tempo total de execução da carga de trabalho.
  - Tempo total de espera de dados por requisições de leitura e escrita.
  - Tempo médio de atendimento das requisições de leitura e escrita.
  - Número de instruções por segundo (MIPS).
  - Número de operações de ponto flutuante por segundo (MFLOPS).
  - Instruções prontas por ciclos de máquina (IPC).
- Métricas Físicas
  - Área ocupada pela memória *cache*.
  - Consumo de potência dinâmica do sistema de memória *cache*.
  - Consumo de potência estática do sistema de memória *cache*.

3.4.3. Projeto de Experimentos

O objetivo de um projeto de experimento (DoE - *Design of Experiment*) correto é obter o máximo de informações com o menor número de experimentos. Além disso, uma correta análise desses experimentos também ajuda a identificar vários fatores ou os fatores que mais influenciam no desempenho.

Alguns tipos de design existem e podem ser considerados para o estudo de desempenho computacional. Alguns deles são [24]:

- *Simple Design*: É considerado um dos mais simples, consiste em variar um fator por vez, e verificar como cada fator influência no desempenho.

- *Full Factorial Design*: Este modelo utiliza todas as combinações de fatores possíveis e faz o mesmo para cada configuração possível.

- *Fractional Factorial Design*: Este modelo utiliza regras de escolha de combinações, a fim de reduzir o número de experimentos, mesmo assim, essa técnica continua gerando um bom nível de detalhamento dos resultados.

O modelo *Fractional Factorial Design* é um dos modelos mais indicados para avaliação de desempenho, principalmente em sistemas simulados onde o número de experimentos deve ser controlado e não muito grande. Dentro deste modelo também existem algumas sub-estratégias de projeto, como os *designs* fatoriais 2^k, 2^{k-p}, e os designs fatoriais com replicação $2^k r$.

Após o estudo sobre os tipos de *designs* de experimentos, foi decidido utilizar o modelo *Fractional Factorial Design* no projeto de avaliação de memórias *cache*, uma vez que esse *design* se molda melhor aos requisitos de simulação e teste pretendidos, e a escolha do sub-modelo foi feita de acordo com cada experimento pretendido, escolhendo o mais adequado às necessidades.

3.5. Simulação de Sistemas Completos

A avaliação de desempenho pode ser classificada em modelagem de desempenho e medição de desempenho [25], a modelagem de desempenho é tipicamente dividida em simulação e modelagem analítica.

Assim, a avaliação de sistemas computacionais [24] pode ser feita de três maneiras diferentes: modelagem analítica, simulação ou medições. A modelagem de desempenho é tipicamente utilizada em estágios anteriores ao processo de projeto, quando os sistemas ainda não estão disponíveis. Desta forma, medições só podem ser efetuadas se algum sistema similar já estiver implementado. Não havendo protótipo disponível, simulações são alternativas viáveis para sistemas completos enquanto modelos analíticos são mais adequados para subsistemas.

Para os sistemas computacionais como subsistemas de memória, onde existem muitas variáveis, a complexidade é muito alta, tornando difícil a criação de modelos analíticos que representem corretamente o sistema. Além disso, modelos analíticos, nesses casos, costumam apresentar baixa precisão.

Assim, a simulação se torna a ferramenta mais apropriada para estimar e comparar características de diferentes organizações e arquiteturas de memórias, mantendo uma boa precisão e boa generalização dos resultados obtidos.

3.6. Utilizando o Simics em Sala de Aula

O ambiente de simulação adotado foi o Simics da Virtutech AB [26], o qual foi escolhido por ser um simulador completo de sistema no nível do conjunto de instruções.

Atualmente para se obter o Simics, é necessária a submissão para a Virtutech e aprovação, de um projeto utilizando o Simics, o que deve ser feito pelo orientador ou professor responsável. O projeto pode ser tanto de pesquisa ou de uso em sala de aula e deve conter informações básicas a respeito de uso e resultados esperados.

Após a submissão e aprovação, a Virtutech deverá entrar em contato liberando o acesso para a download e instalação do Simics e também do servidor de licenças, nesse caso, o servidor de licenças é do tipo flutuante, ou seja, ele trabalha com empréstimos de licenças aos usuários. Uma vez recebido acesso aos recursos, é necessária a instalação do servidor de licenças em algum servidor acessível pelos alunos.

A versão do Simics utilizada neste capítulo é a 4.0, assim, vamos nos limitar a abordar apenas os detalhes relacionados a essa versão. Mesmo assim, são poucas as modificações no ponto de vista do usuário entre as diferentes versões.

Uma das primeiras barreiras que podem aparecer em se tratando de utilização de software com finalidade educacional, é o sistema operacional que o aluno costuma utilizar. Nesse ponto, o Simics possui versões para alguns dos sistemas operacionais mais utilizados: Linux, Windows e Solaris. Dessa forma, a escolha pela ferramenta pode ser feita mais amplamente, pois não apresenta limitações nesse sentido.

A instalação atualmente é feita de maneira bastante intuitiva, portanto não será discutida com muitos detalhes. Porém, sugerimos que no primeiro momento sejam instalados todos os pacotes disponíveis além do pacote básico, tais pacotes dizem respeito à máquina a ser simulada, como, por exemplo, X86, Sparc, Mips, etc. Porém, conforme for adquirida mais experiência, pode-se apenas instalar o pacote específico a ser utilizado.

Depois de efetuada a instalação, faz-se necessária a configuração do *workspace*, nesse ponto o aluno pode criar a área de trabalho no local de maior conveniência. Após a criação do *workspace*, a pasta onde o mesmo foi criado, receberá diversos *links* para os executáveis do Simics e também para as máquinas suportadas pela atual instalação.

Para que os alunos possam utilizar o Simics, é necessário que eles definam em suas máquinas as variáveis de ambiente: *VTECH_LICENSE_FILE* e *LM_LICENSE_FILE* para o valor *{porta}@{servidor_de_licenças}*. Dessa maneira o Simics irá encontrar o caminho para o servidor de licenças.

O simulador suporta a simulação completa de diversos tipos de máquinas e processadores, Alpha, ARM, PPC, IA-64, x86, MIPS, Sparc V8, Sparc V9 e até UltraSparc T1. Entretanto, para o melhor andamento do trabalho é sugerido que os alunos utilizem apenas uma determinada máquina, facilitando assim, a interação entre os alunos e também o trabalho de monitoria.

A escolha pela máquina e arquitetura dos processadores a serem simulados pode ser feita considerando diversos fatores como: Sistema de propósito geral; Suporte a *multi-core*; Suporte a ponto-flutuante; Suporte avançado à memória *cache* e; Suporte a ilimitados núcleos de processamento. Desta maneira, apenas os sistemas Serengeti, SunFire e x86-440BX apresentam as características necessárias esses requisitos. Segue abaixo a descrição das três máquinas mais propícias para suportar as simulações de *multi-core*:

- Serengeti: As classes de servidor Sun Fire 3800 - 6800 são modeladas, oferecendo suporte nativo para até 24 processadores UltraSparc III, UltraSparc III Cu (III+), UltraSparc IV, ou UltraSparc IV+. Essa máquina ainda oferece diversos componentes PCI modelados, porém, suporta apenas sistemas operacionais Solaris, oferecendo *scripts* para instalação do Solaris 8, 9 e 10.

- SunFire: Modela servidores da classe Sun Enterprise 3500 - 6500, com suporte nativo para até 30 processadores UltraSparc II. Os sistemas operacionais Linux e Solaris são suportados por esta máquina que também oferece *scripts* para instalação do Solaris 8, 9 e 10.

- x86-440BX: Pode modelar vários sistemas com processadores x86 ou AMD64 baseado no *chipset* Intel 440BX. Suporta os sistemas operacionais Windows, Linux e NetBSD. A BIOS customizada oferece suporte para até 15 processadores, porém, é importante ressaltar que diversas versões de Linux são limitadas a 8 processadores, enquanto versões de Windows oferecem suportes variados a processadores dependendo da versão do sistema operacional.

Após efetuar testes na máquina x86-440BX, foi verificado que não há condições de executar sistemas operacionais com mais de 8 núcleos de processamento, seja por problemas na compilação do *kernel*, seja por problemas internos no simulador. Além disso, essa máquina com suporte a memória *cache* ativo, apresentou alguns problemas já reportados como *bugs* pela Virtutech. Fora esses problemas, essa máquina apresenta um tempo de simulação superior as duas outras eleita, por esses motivos, atualmente todos os alunos são desencorajados a utilizarem a máquina x86-440BX em suas simulações, até que os problemas citados sejam resolvidos.

Assim, são recomendadas apenas as máquinas SunFire e Serengeti para o uso dos alunos. Aos alunos que necessitem eventualmente de outra máquina mais especifica em seus experimentos, essa deve ser adotada com bastante cautela, observando atentamente as limitações descritas no manual do usuário referente à máquina escolhida.

Algumas limitações do Simics são amplamente conhecidas, as principais são dadas uma vez que o simulador Simics não modela todos os componentes de um *hardware* real, já que o simulador trabalha no nível de conjunto de instruções.

Dessa forma, os resultados de tempo de execução no simulador são dados basicamente em instruções e ciclos. O número de ciclos é dado pelo número de instruções executadas mais os ciclos em espera gerados pelas latências de todos os componentes modelados. Normalmente a execução no simulador, desconsiderando os eventos de alta latência, é de uma instrução por ciclo (parâmetro pré-definido pelo IPC), enquanto uma máquina real consegue gerenciar a execução de mais instruções utilizando superescalaridade e outros componentes de desempenho.

Fora essa limitação, diversos componentes como barramentos, pré-busca e comportamentos do sistema, como gargalos de acesso a recursos, não são totalmente simulados. Desse modo fica claro que o objetivo do simulador Simics é simular máquinas no nível de conjunto de instruções e não efetuar uma simulação exata de todos dispositivos.

Entretanto, o simulador é bastante útil para a comparação de execução de cargas de trabalho em diferentes organizações modeladas dentro do próprio simulador, uma vez que as tendências ocorridas nas simulações devem se repetir em sistemas reais. Logo, o simulador Simics fornece um ambiente controlado e com determinismo controlado, propício para avaliação de sistemas computacional futuros [27].

Algumas dessas limitações podem servir de inspiração aos alunos, como o fato de não ser considerado a contenção de número de portas da memória, por exemplo, pode intrigar o aluno e levá-lo a propor a criação de um modulo para que o Simics modele esse comportamento de gargalo mais corretamente. Além desse exemplo, outros podem ser pensados e sugeridos aos alunos, como forma de incentivo a criação de novos módulos e melhorias na ferramenta.

3.6.1. Modos de Operação

Para simulação de memória *cache*, o Simics provê ferramentas bastante flexíveis para configurar dispositivos. Em uma simulação de memória *cache*, o Simics permite tanto observar informações a respeito de tempo, como a respeito das informações contidas na memória. O simulador possui dois modelos pré-modelados de memórias *cache*: *g-cache* e *g-cache-ooo*:

- O modelo g-cache fornece uma modelagem básica de uma memória *cache* ligada a um processador executando as instruções em ordem e fornecendo relatórios sobre as atividades realizadas.
- O modelo g-cache-ooo além das funcionalidades apresentadas pela g-cache, provê ainda, a possibilidade de ser utilizada em simulações de processamento fora de ordem (OOO - *Out Of Order*).

Mesmo com esses modelos pré-definidos de execução, é importante saber quais dispositivos estão por trás ao executar os dois modelos pré-modelados, uma vez que muitas vezes eses modelos não atendem a todas necessidades do experimento. Por esse motivo, costuma-se não utilizar esses modelos prontos, e sim modelar o sistema e habilitar o modo desejado do simulador.

Existem três modos de simulação aceitos pelo Simics: *Normal*, *Stall* e *MAI*. Esses modos são habilitados na hora de executar o Simics pelas flags *-fast*, *-stall* e *-mai* respectivamente, segue abaixo a descrição de cada um desses modos de operação:

- O modo Normal (*-fast*) fornece um ambiente de simulação onde não são considerados os atrasos referentes à memória, além disso, esse modo não fornece estatísticas de memória *cache* confiáveis uma vez que nem todos os acessos são registrados. Este modo é bastante indicado para a preparação do ambiente de simulação, como por exemplo, instalação de compiladores, compilação e testes preliminares do ambiente de simulação. Além da preparação do ambiente, este modo também pode ser utilizado para simulações onde se requer menor tempo de simulação e não é necessário considerar as latências de memória, como para retirar traços de execução.

- O modo Stall (*-stall*) é um modo de simulação mais lento que o modo normal, uma vez que este modo considera o sistema de memória modelado e também faz o registro de todas as operações ocorridas nas memórias *cache*s. Este modo é indicado para ser utilizado apenas depois que o ambiente de simulação e a máquina a ser simulada estejam totalmente configurados e prontos para os testes.

- O modo MAI (*-mai*) conhecido também como modo micro-arquitetura, é capaz de prover simulação com execução de instruções fora de ordem (OOO). Embora este modo seja bastante atraente, ele não costuma ser muito utilizado devido ao enorme tempo de simulação exigido. Logo, este modo é mais indicado para simulações de curtos trechos de execução, onde se quer analisar o comportamento de apenas um método ou função.

Considerando que o principal objeto de estudo é o sistema de memória *cache*, optou-se por não utilizar o modo MAI, uma vez que este modo levaria ao aumento significativo de complexidade de simulação do processador o que acarretaria a problemas de restrição de tempo, tornando proibitiva a simulação de aplicativos inteiros.

3.6.2. Modelagem de Memórias *Cache*

Um exemplo de modelagem de um processador com dois núcleos de processamento com memórias *cache* de primeiro e segundo nível privadas para cada um dos núcleos é ilustrado na Figura 3.9.

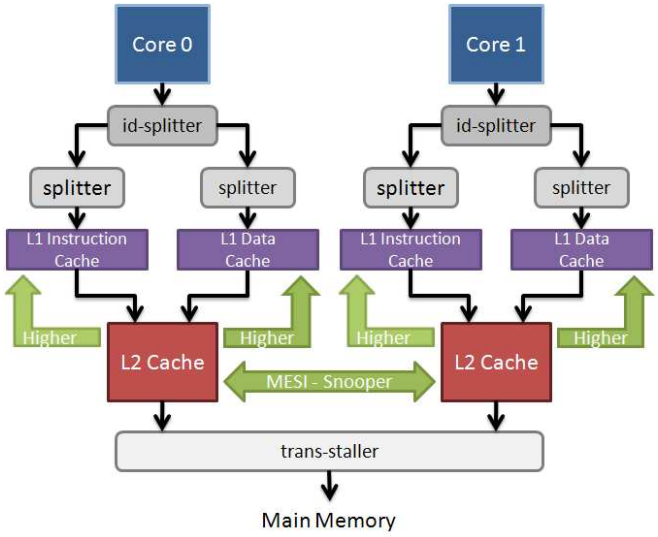


Figura 3.9. Diagrama de modelagem de um *chip multi-core*, adaptado de [28].

Na Figura 3.9 podemos ver os seguintes componentes internos do Simics responsáveis pela correta simulação da memória *cache*:

- Id-splitter - Utilizado para fazer a separação entre instruções e dados para a memória *cache* correta.

- Splitter - Módulo responsável por particionar os dados, a fim de só alocar a quantidade de dados coerente com o tamanho de memória.
- Trans-staller - Dispositivo simples, que simula a latência de acesso da memória principal.

Além dos componentes inerentes à modelagem da memória *cache* no simulador, podemos ver os seguintes componentes referentes ao modelo de controle de coerência de estado das memória *cache* privadas:

- MESI Snooper - Componente definido para indicar quais memórias *cache* de mesmo nível devem utilizar o modo *snooper* a fim de manter coerência.
- Higher Level - Indica ao módulo de coerência do simulador quais memórias *cache*, por exemplo memória *cache* L1, estão em níveis superiores e que se comunicarão diretamente com cada memória *cache* de nível inferior, por exemplo memória *cache* L2.

As interconexões utilizadas no simulador são transparentes, assim, pode-se considerar que as conexões entre os componentes como processador e memória *cache* são ponto a ponto. Entretanto, é possível definir uma latência para as interconexões de memória *cache*, logo, todas as transações que tiverem que passar pela interconexão irá causar latência ao sistema.

Os parâmetros de definição das latências de interconexões no Simics está agregada com cada memória *cache*, essas latências são sempre definidas como latência para acessar o próximo nível, ou seja, partindo do nível mais próximo do núcleo de processamento para o nível inferior da memória.

Seguindo o exemplo de modelagem abaixo apresentado, os experimentos descritos na próxima seção foram modelados, fazendo as devidas modificações na organização e parâmetros das memórias *cache* de acordo com as características a serem estudadas e avaliadas.

```
#=================================================================
## Inicialização de Parâmetros
#=================================================================
$num_cpus = 2
$cpu_class = "ultrasparc-iii-plus"
$clustered = "yes"
run-command-file "%simics%/targets/{machine}"


#=================================================================
## Gerador de Latências da Memória Principal
#=================================================================
@staller0 = pre_conf_object('staller0', 'trans-staller')
@staller0.stall_time = 0


#=================================================================
## Memória Cache L2 - l2c0
#=================================================================
@l2c0 = pre_conf_object('l2c0', 'g-cache')
@l2c0.cpus = [conf.cpu0]
@l2c0.config_line_number = 16384
@l2c0.config_line_size = 64
@l2c0.config_assoc = 8
@l2c0.config_virtual_index = 0
@l2c0.config_virtual_tag = 0
```

```
@l2c0.config_write_back = 1
@l2c0.config_write_allocate = 1
@l2c0.config_replacement_policy = 'lru'
@l2c0.penalty_read = 0
@l2c0.penalty_write = 0
@l2c0.penalty_read_next = 0
@l2c0.penalty_write_next = 0
@l2c0.timing_model = staller0


#=====================================================================
## Memória Cache de Instruções L1 - ic0
#=====================================================================
@ic0 = pre_conf_object('ic0', 'g-cache')
@ic0.cpus = conf.cpu0
@ic0.config_line_number = 1024
@ic0.config_line_size = 32
@ic0.config_assoc = 2
@ic0.config_virtual_index = 0
@ic0.config_virtual_tag = 0
@ic0.config_write_back = 0
@ic0.config_write_allocate = 0
@ic0.config_replacement_policy = 'lru'
@ic0.penalty_read = 0
@ic0.penalty_write = 0
@ic0.penalty_read_next = 0
@ic0.penalty_write_next = 0
@ic0.timing_model = l2c0


#=====================================================================
## Memória Cache de Dados L1 - id0
#=====================================================================
@dc0 = pre_conf_object('dc0', 'g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 1024
@dc0.config_line_size = 32
@dc0.config_assoc = 2
@dc0.config_virtual_index = 0
@dc0.config_virtual_tag = 0
@dc0.config_write_back = 0
@dc0.config_write_allocate = 0
@dc0.config_replacement_policy = 'lru'
@dc0.penalty_read = 0
@dc0.penalty_write = 0
@dc0.penalty_read_next = 0
@dc0.penalty_write_next = 0
@dc0.timing_model = l2c0


#=====================================================================
## Transaction Splitter para Memória Cache L1 de Instruções
#=====================================================================
@ts_i0 = pre_conf_object('ts_i0', 'trans-splitter')
@ts_i0.cache = ic0
@ts_i0.timing_model = ic0
@ts_i0.next_cache_line_size = 64


#=====================================================================
## Transaction Splitter  para Memória Cache L1 de Dados
#=====================================================================
@ts_d0 = pre_conf_object('ts_d0', 'trans-splitter')
@ts_d0.cache = dc0
@ts_d0.timing_model = dc0
@ts_d0.next_cache_line_size = 64


#=====================================================================
## ID Splitter para Memória Cache L1
#=====================================================================
@id0 = pre_conf_object('id0', 'id-splitter')
@id0.ibranch = ts_i0
@id0.dbranch = ts_d0


#=====================================================================
## Memória Cache L2 - l2c1
#=====================================================================
@l2c1 = pre_conf_object('l2c1', 'g-cache')
@l2c1.cpus = [conf.cpu1]
@l2c1.config_line_number = 16384
@l2c1.config_line_size = 64
@l2c1.config_assoc = 8
@l2c1.config_virtual_index = 0
@l2c1.config_virtual_tag = 0
@l2c1.config_write_back = 1
@l2c1.config_write_allocate = 1
@l2c1.config_replacement_policy = 'lru'
@l2c1.penalty_read = 0
@l2c1.penalty_write = 0
@l2c1.penalty_read_next = 0
@l2c1.penalty_write_next = 0
@l2c1.timing_model = staller0
```

```
#============================================================
## Memória Cache de Instruções L1 - ic1
#============================================================
@ic1 = pre_conf_object('ic1', 'g-cache')
@ic1.cpus = conf.cpu1
@ic1.config_line_number = 1024
@ic1.config_line_size = 32
@ic1.config_assoc = 2
@ic1.config_virtual_index = 0
@ic1.config_virtual_tag = 0
@ic1.config_write_back = 0
@ic1.config_write_allocate = 0
@ic1.config_replacement_policy = 'lru'
@ic1.penalty_read = 0
@ic1.penalty_write = 0
@ic1.penalty_read_next = 0
@ic1.penalty_write_next = 0
@ic1.timing_model = l2c1


#============================================================
## Memória Cache de Dados L1 - id1
#============================================================
@dc1 = pre_conf_object('dc1', 'g-cache')
@dc1.cpus = conf.cpu1
@dc1.config_line_number = 1024
@dc1.config_line_size = 32
@dc1.config_assoc = 2
@dc1.config_virtual_index = 0
@dc1.config_virtual_tag = 0
@dc1.config_write_back = 0
@dc1.config_write_allocate = 0
@dc1.config_replacement_policy = 'lru'
@dc1.penalty_read = 0
@dc1.penalty_write = 0
@dc1.penalty_read_next = 0
@dc1.penalty_write_next = 0
@dc1.timing_model = l2c1


#============================================================
## Transaction Splitter para Memória Cache L1 de Instruções
#============================================================
@ts_i1 = pre_conf_object('ts_i1', 'trans-splitter')
@ts_i1.cache = ic1
@ts_i1.timing_model = ic1
@ts_i1.next_cache_line_size = 64


#============================================================
## Transaction Splitter para Memória Cache L1 de Dados
#============================================================
@ts_d1 = pre_conf_object('ts_d1', 'trans-splitter')
@ts_d1.cache = dc1
@ts_d1.timing_model = dc1
@ts_d1.next_cache_line_size = 64


#============================================================
## ID Splitter  para Memória Cache L1
#============================================================
@id1 = pre_conf_object('id1', 'id-splitter')
@id1.ibranch = ts_i1
@id1.dbranch = ts_d1


#============================================================
# Níveis Superiores de Memória Cache
#============================================================
@l2c0.higher_level_caches = [ic0,dc0]
@l2c1.higher_level_caches = [ic1,dc1]


#============================================================
# Snoopers
#============================================================
@l2c0.snoopers = [l2c1]
@l2c1.snoopers = [l2c0]


#============================================================
# Criação de Espaços de Memória para os Núcleos
#============================================================
@mem0 = pre_conf_object('cpu0_space','memory-space')
@mem1 = pre_conf_object('cpu1_space','memory-space')


#============================================================
# Instanciação dos Componentes
#============================================================
@SIM_add_configuration([staller0,l2c0,mem0,ic0,dc0,ts_i0,ts_d0,id0,l2c1,mem1,ic1,dc1,ts_i1,ts_d1,id1], None);
```

```
#=============================================================
# Ligação dos Modelos de Tempo
#=============================================================
@conf.cpu0_space.default_target = [conf.phys_mem, 0, 0, conf.phys_mem]
@conf.cpu0.physical_memory = conf.cpu0_space
@conf.cpu0_space.timing_model = conf.id0
@conf.cpu1_space.default_target = [conf.phys_mem, 0, 0, conf.phys_mem]
@conf.cpu1.physical_memory = conf.cpu1_space
@conf.cpu1_space.timing_model = conf.id1
#=============================================================
```

Como o simulador trabalha em nível de conjunto de instruções, as penalidades para os diversos componentes simulados são definidos em termos de ciclo de relógio. Desta maneira, uma latência de leitura de dados deve ser modelada em termos de ciclos, considerando a freqüência que se deseja modelar e o tempo de acesso.

Uma importante ferramenta para estimar parâmetros de memória *cache* e memória principal é o *software* Cacti [29]. Assim, pode-se modelar todas as estimativas de tempo de acesso às memórias considerando as estimativas de latência fornecidas por essa ferramenta. Para modelar uma memória corretamente no Cacti, deve-se fornecer informações como a tecnologia de integração, freqüência de operação e outros parâmetros os quais devem ser definidos previamente para os experimentos e podem ser desenvolvidos em sala de aula com os alunos.

3.6.3. Discos, Imagens e Checkpoints

Ao instalar os pacotes do Simics, pode-se notar que diversas máquinas não funcionam corretamente por falta da imagem do disco rígido. Isso acontece, pois essas imagens costumam ser bastante grandes e por isso a Virtutech disponibiliza os arquivos de imagem separados, apenas na seção de downloads do site *www.simics.net*.

Dessa maneira, ao fornecer a ferramenta de simulação aos alunos, é importante fornecer também informações sobre como conseguir essas imagens de discos que já possuem um sistema operacional instalado. Assim, os alunos poderão avançar mais rapidamente para a parte de modelagem e simulação, diminuindo assim possíveis atrasos que essa etapa de instalação e configuração possa causar ao projeto.

Seguindo o exemplo da Virtutech, que disponibiliza sistemas pré-configurados para as máquinas a serem simuladas. Os professores ou monitores da disciplina podem deixar imagens de sistemas prontos com as ferramentas básicas necessárias aos alunos, para que essa etapa não consuma muito tempo do projeto.

Para a criação dessas máquinas pré-configuradas, existem dois tipos de arquivos básicos de imagem de discos rígidos no Simics, os arquivos do tipo *craff* e os arquivos do tipo *disk*. Esses dois tipos de arquivos são gerados pelo próprio simulador, e podem ser gerados para qualquer sistema configurado, de forma que os alunos podem apenas obter esses arquivos e conectar em suas simulações.

Os arquivos do tipo *craff*, costumam ser de pacotes mais consolidados como os cedidos pela Virtutech para download e devem ser colocados em pasta especifica dentro da instalação do Simics. Já as instalações feitas pelo usuário costumam ser do tipo *disk* e essas precisam apenas ser copiadas para dentro do *workspace* do usuário, para que comecem a funcionar.

Além desses dois tipos de arquivos de imagem, o aluno pode ainda gerar cópias de segurança durante a simulação, seja do estado persistente do disco rígido ou de todo contexto simulado, registradores, disco, memória, etc.

Para a gravação de estados persistentes, é importante que o disco a ser salvo esteja sincronizado, ou seja, todos os dados devem sofrer um *write-back* para o disco rígido, e então se pode utilizar os comandos, *save-persistent-state* para gravar ou *load-persistent-state*, para carregar o estado do disco rígido da máquina simulada. No Linux por exemplo, o aluno pode fazer a sincronização como o comando *sync*, ou então desligando a máquina simulada.

Para a criação de *checkpoints* das máquinas simuladas, com todos dados da execução, deve-se utilizar os comandos *write-configuration* para salvar e *read-configuration* para carregar o ponto de restauração salvo. No entanto, esse modo de salvamento não permite que a arquitetura seja modificada, pois isso poderia criar inconsistências na simulação.

Como esse recurso de criação de estados persistentes, a preparação do ambiente pode ser feita em uma simulação simples, diferente da modelagem da máquina final. Dessa forma, a simulação tende ser mais veloz, e então, somente ao final da preparação do ambiente deve-se criar um estado persistente o qual irá manter as modificações no disco rígido simulado.

Após a completa configuração do sistema, chega a hora de modelar a arquitetura que for adequada para os experimentos e carregar o estado persistente previamente criado. Assim, ao final da carga do sistema ou durante os testes, o usuário também pode criar eventuais pontos de checagem como forma de *backup* ou até mesmo auxílio durante a simulação.

3.6.4. Inserindo Arquivos no Ambiente Simulado

Uma das funções básicas para uma simulação completa é a inserção de arquivos no ambiente simulado. O simulador Simics permite a inserção de arquivos de três maneiras distintas, por módulo de *kernel*, rede local ou imagem de CD-ROM. Os estados persistentes de discos rígidos disponibilizados pela Virtutech possuem uma preparação especial do sistema operacional, onde o módulo de *kernel* chamado SimicsFS carregado ao *kernel* permite que dentro da simulação o usuário possa montar o sistema de arquivos da máquina hospedeiro, ou seja, onde se encontra instalado o Simics.

Mesmo com a facilidade que um módulo já incorporado ao *kernel* possa propiciar, ao montar um ambiente de simulação começando desde a instalação do sistema operacional, a tarefa de compilar o *kernel* com o módulo SimicsFS pode não ser muito fácil. Assim, o usuário ainda conta com duas alternativas para carregar os arquivos para dentro da simulação, ou habilitando a rede do hóspede (máquina que está sendo simulada) e ativando o suporte do Simics para conexão com a rede real. Ou então, ainda podemos contar com a inserção dos arquivos pela simulação de uma unidade de CD-ROM na máquina simulada, para isso, basta criar um arquivo de imagem do tipo ISO (*mkisofs -l -o New-Image.iso -r Files-to-image*), e então utilizar os comandos, (*new-file-cdrom file = New-Image.iso*), para criar um CD virtual com o arquivo *iso* e então o comando (*cd0.insert*

media = New-Image) para inserir o CD virtual no dispositivo de CD-ROM da máquina simulada.

3.6.5. *Magic Instruction*

O Simics fornece uma maneira de interação entre simulação e simulador bastante simples e eficiente, conhecida como *Magic Instruction*. A *Magic Instruction* nada mais é que uma biblioteca com as funções MAGIC e MAGIC_BREAKPOINT, essas funções quando chamadas, elas executam um código em linguagem de montagem que não modifica nenhum registrador, mas é reconhecido pelo simulador como sendo a chamada da *Magic Instruction* e então efetua a tarefa indicada.

Um código simples de HelloWorld.c em OpenMP com a chamada para a *Magic Instruction* segue abaixo:

```
//------------------HelloWorld.c------------------\\

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include "magic-instruction.h"

int main (int argc, char *argv[]) {
 int t_id;

 MAGIC(1);
 #ifdef _OPENMP
   printf("Executando: OpenMP -> %d threads\n",
                            omp_get_max_threads());
 #else
   printf("Executando: Normal -> 1 thread\n");
 #endif

 #ifdef _OPENMP
 #pragma omp parallel private(t_id)
   {
 #endif
    t_id = omp_get_thread_num();
    printf("Oi mundo da thread %d\n", t_id);
 #ifdef _OPENMP
   }  /* Sincronizacao */
 #endif

 printf("Isso eh impresso por apenas uma thread.\n");
 MAGIC_BREAKPOINT;
}
```

No código *HelloWorld.c*, é importante notar que foi incluída a biblioteca *magic-instruction.h* no inicio do código, e então foram utilizadas as chamadas *MAGIC(1)* e *MAGIC_BREAKPOINT*. A função *MAGIC(1)* irá retornar para o Simics e executar se disponível alguma função de *callback* pré-definida. Já a função *MAGIC_BREAKPOINT* faz com que a simulação pare ao executar essa função. Dessa maneira, com essas funções a gama de possibilidades de uso do simulador aumenta bastante, uma vez que o usuário pode instrumentar qualquer código a fim de interagir com o sistema simulado. Ressaltando que a biblioteca *magic-instruction.h* está contida no pacote de instalação do Simics.

Abaixo segue a definição da função em Python no Simics para ser executada durante o *callback* da função *MAGIC()*:

```
@def a_callback(user_arg, cpu, arg):
        print "numero de ciclos", cpu.cycles
        SIM_run_command_file("FuncaoA.simics")
        @SIM_hap_add_callback("Core_Magic_Instruction", a_callback, None)
```

A função de *callback* recebe a chamada da *Magic Instruction* e então executa os comando do arquivo *FuncaoA.simics* que pode conter funções em Python ou CLI.

3.6.6. Modo de Busca de Instruções

Algumas configurações são necessárias em se tratando de simulação de memórias *cache* a fim de conseguir resultados coerentes e mais próximos da realidade. Nesse sentido, a busca de instruções no simics pode ser habilitada ou desabilitada pelo usuário de acordo com suas necessidades de simulação.

Tal configuração é feita pelo comando *ifm* (*instruction fetch mode*), esse comando seleciona como as buscas de instrução serão enviadas para a hierarquia de memória durante a simulação. Se configurado para *no-instruction-fetch*, a hierarquia de memória não irá receber nenhuma busca de instruções durante a simulação. Caso configurado para *instruction-cache-access-trace*, a hierarquia de memória irá receber uma e apenas uma busca de para cada nova linha de dados acessada, onde o tamanho dessa linha da memória *cache* é definida pelo atributo *instruction-fetch-line-size* dentro do processador. Por fim, caso a busca de instruções for configurada para *instruction-fetch-trace*, todas as buscas de instrução serão visíveis.

Mesmo esse comando sendo bastante interessante, nem todos os níveis de configuração citados estão disponíveis em todas as arquiteturas simuladas, devendo ser consultado o manual para cada arquitetura a fim de saber a disponibilidade.

Uma vez que os atuais processadores utilizam mecanismos de *pre-fetch*, é importante ponderar o uso do modo de busca de instruções, onde por um lado pode ser mais realístico o registro de busca apenas quando uma nova linha de memória é acessada, ou por outro lado, o registro de todas as buscas, com latências reduzidas podem gerar melhores aproximações. Assim, é importante a discussão em sala de aula, sobre os modos de busca de dados, e qual o modo deverá ser adotado a fim de gerar uma simulação mais realística e envolver os alunos no tema, gerando um pensamento mais crítico sobre as arquiteturas e seus mecanismos.

3.6.7. Métricas e configurações de execução

Uma vez que o Simics trabalha em nível de conjunto de instruções, o simulador não é capaz de indicar qual será o IPC da máquina simulada, uma vez que os estágios de um pipeline, por exemplo, não são simulados. Porém, o simulador permite que sejam configurados diversos mecanismos a respeito da execução durante a simulação.

A primeira configuração necessária para uma simulação mais realista é o tempo de troca de execução entre processadores/núcleos. Essa configuração existe por causa da forma de implementação do simulador, e pode influenciar o tempo de simulação.

O Simics, trabalha com múltiplos processadores da seguinte forma. Tomando o caso hipotético de 4 processadores sendo simulados, o Simics irá executar N ciclos do processador 0, N ciclos do processador 1, e assim por diante, em um esquema de fila circular. Para simulações onde não estamos interessados exatamente no efeito das memórias *cache*, ou o tempo exato de simulação não é importante, o número de passos executados N pode ser grande. Já nos casos onde queremos ter resultados mais precisos, deve-se usar N = 1.

O comando para definir o tempo dessa troca de execução é o *cpu-switch-time cycles*, o qual é indicado utilizar valor igual a 1 apenas quando estamos com todo ambiente

de simulação pronto, e queremos fazer testes em nossa arquitetura. Dessa maneira, com cada processador executando 1 ciclo por vez, a simulação pode gerar resultados mais próximos da realidade, porém essa troca de execução é custosa no ponto de vista de tempo de simulação.

Após a configuração do tempo de troca de execução entre processadores, outro parâmetro importante que pode ser configurado no Simics é a quantidade de instruções executadas por ciclo (IPC) de cada processador. Essa configuração pode ser feita pelo método *set-step-rate* de cada processador, onde a configuração padrão é igual a 1. Com esse parâmetro, pode-se executar uma dada aplicação em uma máquina real, obter o seu IPC e então configurar no Simics, fazendo uma configuração fina para a simulação. Essa configuração também pode ser utilizada em inúmeras experiências, como por exemplo, simulação de núcleos heterogêneos, onde um ou mais núcleos tem uma capacidade de computação diferente dos demais.

Para suportar essas configurações de IPC, três conceitos são utilizados para simulação: *step*, *cycles* e *time*. O entendimento desses conceitos é fundamental para alunos de arquiteturas, e pode ser exercitado utilizando o simulador.

O número de *steps* nada mais é que o número de instruções completadas ou exceções ocorridas. Ou seja, são as instruções completadas somadas com as instruções que não foram completadas por que alguma exceção ocorreu. Onde cada instrução irá contar como um *step*.

A quantidade de *cycles* é o número de ciclos executados desde que a máquina simulada foi ligada. A quantidade de *cycles* e *steps* não são necessariamente iguais, uma vez que uma instrução pode gerar eventos de alta latência como acesso à memória, além do que esse número pode ser influenciado também pelo IPC configurado para o processador.

O *time* dentro do simulador não é influenciado pela máquina em que a simulação está sendo executada (máquina hospedeira). De forma que hospede e hospedeiro são totalmente independentes. Para a atualização do tempo dentro da simulação, é utilizada a quantidade de *cycles* dividida pela freqüência definida para o processador. Essa independência é importante para que o simulador se mantenha determinístico, e assim, independente da carga de trabalho que a máquina hospedeira esteja executando, o simulador deverá retornar sempre resultados coerentes da simulação.

Considerando o isolamento da máquina simulada, podemos notar que os resultados de desempenho de uma simulação podem ser vistos tanto dentro do simulador, como se estivesse em um sistema real, o então o aluno pode utilizar as *Magic Instruction* dentro do código a ser executado e então obter o número exato de ciclos de execução. A segunda alternativa, efetuando a instrumentação do código com *Magic Instruction*, é considerado o modo mais preciso e interessante para que sejam feitas medições de desempenho. Uma vez que, uilizando *Magic Instruction*, pode-se além de automatizar o processo de geração de estatísticas e relatórios, esse modo permite também obter mais resultados, como as estatísticas da memória *cache* do simulador.

As estatísticas da memória *cache* disponíveis no Simics são obtidas por métodos que podem ser executados em qualquer objeto de memória *cache* no Simics, abaixo segue a lista e breve descrição das principais estatísticas disponíveis:

- stat_copy_back - Número de transações de *copy-back* iniciadas pela memória *cache*.
- stat_data_read - Número de leituras de dados na memória *cache*
- stat_data_read_miss - Número de leituras de dados que não estavam na memória *cache*.
- stat_data_write - Número de escritas de dados na memória *cache*
- stat_data_write_miss - Número de escrita de dados que não estavam na memória *cache*.
- stat_inst_fetch - Número de busca de instruções na memória *cache*.
- stat_inst_fetch_miss - Número de busca de instruções que não estavam na memória *cache*.
- stat_mesi_exclusive_to_shared - Número de operações do protocolo MESI do tipo: Exclusivo para Compartilhado.
- stat_mesi_invalidate - Número de operações de invalidações de dados do protocolo MESI.
- stat_mesi_modified_to_shared - Número de operações do protocolo MESI do tipo: Modificado para Compartilhado.
- stat_transaction - Número total de transações observadas pela memória *cache*.

3.6.8. Determinismo nas Simulações

A etapa de resultados de uma avaliação de desempenho é uma das mais importantes, uma vez que todos os dados e as análises feitas acerca dos dados devem ser compilados e apresentados de forma clara e objetiva. Justamente nesta etapa é que ocorrem muitos erros ligados à avaliação do sistema computacional [24], como ignorar a variação das medidas, apresentação dos resultados de forma incorreta, assim como omitir as limitações do sistema.

Por isso, quando estamos trabalhando em sistemas reais, normalmente são feitas múltiplas medições, e técnicas estatísticas são utilizadas para garantir que as medições são estatisticamente significantes. Logo, o objetivo dessa metodologia é garantir que efeitos causados por fatores não controlados, não levem a falsas conclusões.

Assim, ao trabalhar com um simulador determinístico, onde todas as execuções de uma mesma aplicação iniciadas de um mesmo *checkpoint* vão retornar os mesmos resultados, uma vez que o simulador irá executar as mesmas operações e sofrer de mesmas latências do *hardware*, como é o caso do Simics, poderíamos considerar que temos um resultado plausível a partir de apenas uma medição. Entretanto, devemos considerar que o fato do simulador ser determinístico não garante que a carga de trabalho também será determinística. Ou seja, pequenas variações iniciais do estado do simulador podem levar a carga de trabalho ou o sistema operacional a caminhos de execução diferentes [30], dessa maneira, a execução única do experimento pode levar a falsas conclusões.

Logo, certos cuidados devem ser tomados na simulação com relação à variação dos resultados obtidos, uma vez que o simulador que estamos utilizando é determinístico. Para inserção de não-determinismo entre as execuções, cada aplicação pode, por exemplo, ser executada a partir de diferentes *checkpoints*, fazendo assim o sistema agir de forma não-determinística, pelas interrupções causadas pelo sistema operacional, levando assim, as cargas de trabalho a diferentes caminhos de execução. Para essa forma de execução, podemos após cada execução de uma determinada aplicação, criar um *checkpoint* diferente, a partir do qual a próxima execução e medição daquela mesma aplicação será feita.

Além da preocupação em relação ao determinismo do sistema, pode-se também planejar reduzir possíveis efeitos transientes [31] executando previamente cada aplicação da carga de trabalho, a fim de aquecer a memória *cache*, salvando o estado do sistema após essa primeira execução de cada aplicação. Então, somente a partir desse estado salvo, executa-se e mede-se cada aplicação da carga de trabalho.

3.6.9. Fluxo de Projeto Proposto

Após apresentar os conceitos e funcionalidades do simulador, é importante fixar um fluxo de projeto para que os alunos possam se guiar durante a execução do projeto proposto.

Assim, com a experiência ganha após alguns testes e avaliações com o Simics, foi desenvolvido o fluxo de projeto apresentado na Figura 3.10.

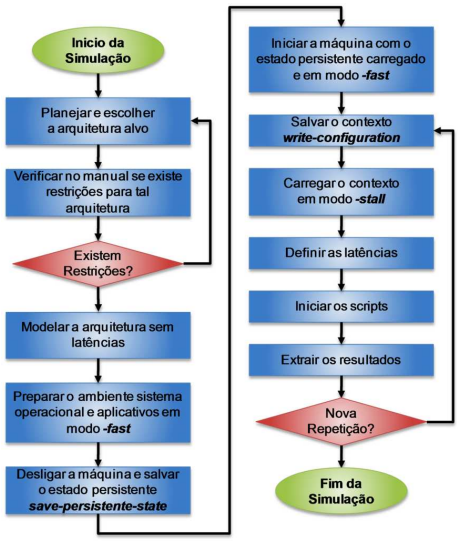


Figura 3.10. Diagrama do fluxo de simulação proposto.

O fluxo de projeto apresentado não considera as etapas de *design* de experimentos nem de análise estatísticas, sendo que essas etapas podem ser agregadas aos projetos, na forma que mais for conveniente ao professor. Porém, as etapas apresentadas no diagrama, auxiliam os alunos no sentido de viasualizar toda a organização do método de trabalho utilizando o simulador, de forma que, utilizando os passos apresentados, os alunos tendem a reduzir o tempo total do projeto.

3.7. Estudo de Casos

Nesta seção será apresentado um estudo de caso de avaliação de memórias *cache* em *multi-core* utilizando o Simics, que serviu de experiência para a adoção do simulador no ensino de arquiteturas. Também serão mostrados como exemplo, seis trabalhos apresentados pelos alunos, dividido pelos dois anos ministrados da disciplina chamada Introdução ao Processamento Paralelo e Distribuído, da Pós-Graduação em Computação da Universidade Federal do Rio Grande do Sul.

Com a contínua demanda por desempenho computacional, as memórias *cache* vêm sendo largamente adotadas nos diversos tipos de projetos arquiteturais de computadores. Os atuais processadores disponíveis no mercado apontam na direção do uso de memórias *cache* L2 compartilhadas. No entanto, ainda não esta claro quais os ganhos e custos inerentes desses modelos de compartilhamento da memória *cache*. Assim, nota-se a importância de estudos que abordem os diversos aspectos do compartilhamento de memória *cache* em processadores com múltiplos núcleos.

Portanto, esse estudo de caso visou avaliar diferentes compartilhamentos de memória *cache*, modelando e aplicando cargas de trabalho sobre as diferentes organizações, a fim de obter resultados significativos sobre o desempenho e a influência do compartilhamento da memória *cache* em processadores *multi-core*.

Para isso, foram avaliados diversos compartilhamentos de memória *cache*, utilizando o simulador Simics, avaliando técnicas tradicionais de aumento de desempenho, como aumento da associatividade, maior tamanho de linha, maior tamanho de memória *cache* e também aumento no número de níveis de memória *cache*, investigando a correlação entre essas arquiteturas de memória *cache* e os diversos tipos de aplicações da carga de trabalho avaliada.

Os resultados mostraram a importância da integração entre os projetos de arquitetura de memória *cache* e o projeto físico da memória, a fim de obter o melhor equilíbrio entre tempo de acesso à memória *cache* e redução de faltas de dados. Notou-se nos resultados, dentro do espaço de projeto avaliado, que devido às limitações físicas e de desempenho, as organizações 1Core/L2 e 2Cores/L2, com tamanho total igual a 32 MB (bancos de 2 MB compartilhados), tamanho de linha igual a 128 *bytes*, representam uma boa escolha de implementação física em sistemas de propósito geral, obtendo um bom desempenho em todas aplicações avaliadas sem grandes sobrecustos de ocupação de área e consumo de energia.

Além disso, como conclusão deste estudo de caso, mostrou-se que, para as atuais e futuras tecnologias de integração, as tradicionais técnicas de ganho de desempenho obtidas com modificações na memória *cache*, como aumento do tamanho das memórias, incremento da associatividade, maiores tamanhos da linha, etc. não devem apresentar ganhos reais de desempenho caso o acréscimo de latência gerado por essas técnicas não seja reduzido, a fim de equilibrar entre a redução na taxa de faltas de dados e o tempo de acesso aos dados.

Nos trabalhos desenvolvidos pelos alunos apresentados a seguir, foi utilizado o simulador Simics, simulando sistemas operacionais Linux ou Solaris com compiladores GNU GCC ou Sun Studio respectivamente.

Inicialmente os alunos utilizaram arquiteturas de processadores *dual-core* e *quad-core* conhecidas, sem efetuar mudanças que representassem alterações ou uma nova proposta de arquitetura. As configurações principais de três trabalhos feitos no primeiro ano que a disciplina adotou o Simics para auxilio no ensino são descritas a seguir:

Um primeiro trabalho, do primeiro ano modelou um processador *quad-core* AMD Opteron de 64 bits. As memórias *cache* L1 são locais a cada núcleo, possuem tamanho de 128 kB cada e não são privadas. A memória *cache* L2 possui tamanho de 512 kB e é compartilhada entre todos os núcleos. Os ciclos de penalidade para cada memória são: 3 ciclos para memória *cache* L1, 10 ciclos para memória *cache* L2 e 60 ciclos para a memória principal. Além da modelagem no simulador, foi desenvolvido um programa paralelo em OpenMP para decodificação de MP3.

Em um segundo trabalho, do primeiro ano foi modelado um processador *dual-core* Pentium 4 de 2GHz. Onde cada núcleo possui uma memória *cache* L1 de 64 kB e os dois núcleos compartilham uma memória *cache* L2 de 2 MB. Foi utilizado uma latência de 3 ciclos para a memória *cache* L1 e 14 ciclos para a memória *cache* L2. Neste trabalho o aluno estudou a organização do Intel Core Duo, núcleo Yonah, para especificar as latências do projeto. A aplicação avaliada nesta arquitetura foi um decodificador MPEG paralelizado através de MPI.

Um terceiro trabalho, do primeiro ano foi focado na arquitetura UltraSparc II. O aluno modelou um processador com quatro núcleos, mantendo também a tendência mostrada nos trabalhos anteriores de compartilhamento de memória *cache* L2 e memórias *cache* L1 locais e privadas. No entanto o aluno desenvolveu dois modelos, no primeiro a memória *cache* L2 é compartilhada entre todos os quatro núcleos, no segundo a memória *cache* L2 é dividida em duas, sendo que apenas dois núcleos compartilham cada uma delas. No primeiro modelo a memória *cache* L1 possui 32 kB e a memória *cache* L2 possui 1024 kB. No segundo modelo a memória *cache* L1 permanece com o mesmo tamanho, mas as memórias *cache* L2 possuem agora 512 kB cada uma. Neste trabalho, o aluno desenvolveu e avaliou um programa para convolução de imagens escrito em MPI.

Ao final do primeiro ano, notamos que o projeto da arquitetura dos processadores poderia ser mais bem elaborado pelos alunos nos próximos semestres, uma vez que a proposta da disciplina deixaria de ser uma novidade fazendo com que os próximos alunos tenham uma real noção do grau de dificuldade e do tempo necessário para realização do projeto. Além disto, trabalhos com o grau de dificuldade e/ou profundidade já apresentados deveriam ser melhorados. Portanto, seria solicitado e esperado, que os próximos trabalhos fossem mais complexos.

A introdução ao Simics e o suporte dado aos alunos se mostrou eficiente. No entanto, percebeu-se a necessidade de introduzir o ambiente de simulação logo no início da disciplina, mesmo antes do ensino teórico sobre arquiteturas paralelas, o que foi feito no segundo ano adotando a ferramenta de simulação. A atividade de monitoria também foi mais incentivada, para que os alunos pudessem tirar dúvidas mais rapidamente.

Assim, no segundo ano da disciplina, o método de ensino do simulador evoluiu, onde mais materiais didáticos foram preparados e algumas formas de apresentar o simulador e exigir projetos também evoluíram. Neste segundo ano, também foi aperfeiçoado o

ambiente a ser disponibilizado aos alunos, parte foi feito previamente, e partes feitas sob demanda, de forma a gerar um processo de melhorias continuas ao longo dos anos usando a ferramenta.

Além disso, nesse segundo ano, foram indicadas duas aplicações CG e BT da carga de trabalho paralela NAS para que os alunos pudessem avaliar as arquiteturas apresentadas. Além disso, com a escolha de uma mesma carga de trabalho para todos os alunos efetuarem testes, planeja-se criar um espírito competitivo a cada ano, a fim de motivar os alunos a criarem propostas cada vez mais interessantes.

A seguir são apresentados três novos trabalhos feitos pelos alunos no segundo ano em que o Simics foi adotado como ferramenta didática na disciplina.

Um primeiro trabalho, do segundo ano avaliou a diferença de desempenho de arquiteturas de três e quatro níveis de memória *cache*, e propôs uma hierarquia entrelaçada de memória. O aluno verificou que, nas situações simuladas, é mais interessante adicionar um quarto nível de memória *cache* e dividir o terceiro em dois módulos, do que simplesmente aumentar o tamanho da memória *cache* L3. Notando que o comportamento se deve ao menor tempo de acesso e a menor contenção pelo acesso a L3 presente nas arquiteturas com memória *cache* L4 simuladas.

Em um segundo trabalho, do segundo ano o aluno teve como objetivo, avaliar a influência de se adicionar um nodo gargalo em um sistema de processamento distribuído, no caso, o aluno criou uma pequena rede particular com 4 nodos simulados, onde foram avaliadas diferentes configurações de nodos com diferentes capacidades de processamento. Os resultados, diferente do esperado não mostraram perdas de desempenho ao adicionar o nodo gargalo na pequena rede, mesmo assim, o aluno usou esses resultados como motivação para criar uma nova metodologia de avaliação para atacar novamente o problema.

Um terceiro trabalho, do segundo ano apresentou diversas simulações de arquiteturas de *multi-core*, com 1, 2, 8 e 16 núcleos de processamento. Onde o aluno motivado por outros artigos que diziam que poderia haver queda de desempenho com 16 núcleos, resolveu apresentar um trabalho avaliando o ganho de desempenho conforme se aumentou o número de núcleos de processamento. Nessa simulação, diferentemente do que foi simulado no artigo base, o aluno não simulou a contenção na memória, que seria gerado, por exemplo, por conflitos de acesso a um número restrito de portas de acesso. Como resultado, não houve quedas de desempenho conforme aumentou a quantidade de núcleos de processamento, levantando a hipótese sobre a influência da contenção de memória, a qual deve ter grande influência no desempenho final do sistema *multi-core*.

O segundo ano da disciplina com o Simics foi positivo, entretanto, houve um contratempo inicial relacionado com as licenças do Simics, uma vez que a Virtutech estava com algumas mudanças no fornecimento de licenças, houve um atraso inicial para que os alunos pudessem ter acesso a ferramenta de simulação.

Porém, a exigência logo no inicio do semestre de projetos bem descritos, com objetivos claros facilitou a execução dos trabalhos pelos alunos, permitindo também uma maior interação com sugestões e criticas na etapa de proposta, fazendo os alunos pensar sobre as metas reais e resultados esperados, de maneira a fortalecer também os conceitos

Atividades	Jan	Fev	Mar	Abr	Mai	Jun	Jul
Pedido de Licenças	X						
Preparação de Discos Virtuais		X	X				
Introdução à Disciplina			X				
Apresentação Teórica do Simics			X				
Definição do Projeto aos Alunos			X				
Apresentação Prática do Simics			X				
Data Limite da Entrega dos Projetos				X			
Apresentação de Andamento dos Trabalhos					X		X
Apresentação Final dos Trabalhos							X
Atividade de Monitoria			X	X	X	X	X

Tabela 3.1. Cronograma de Atividades

de metodologia científica.

A introdução ao Simics e o suporte dado aos alunos se mostrou mais eficiente nesse segundo ano. No entanto, percebeu-se a necessidade de introduzir o ambiente de simulação ainda mais cedo, logo no início da disciplina, mesmo antes do ensino teórico sobre arquiteturas paralelas. Dessa forma, os alunos terão uma noção mais sólida do simulador e suas funcionalidades, facilitando a proposta de projetos por parte dos alunos.

O uso de uma única carga de trabalho por todos os alunos não foi suficiente criar ambiente competitivo entre os alunos, porém, a disponibilização dessas aplicações paralelas, antes mesmo dos alunos aprenderem conceitos de programação paralela, foi de grande importância para o bom andamento dos trabalhos. Uma vez que, os alunos puderam testar suas arquiteturas e apresentar resultados de desempenho durante um seminário de andamento. Com isso os alunos ficaram de certa forma, mais nivelados, evoluindo no projeto da disciplina de forma mais tranqüila.

Mostramos o cronograma sugerido para a utilização do simulador em sala de aula na Tabela 3.1. Como pode ser visto no cronograma, existem duas etapas antes mesmo do início das aulas, que ocorre no mês de Março, essas etapas são planejadas para dar mais tempo de preparação do ambiente de simulação para os alunos. Depois, podemos ver 6 etapas planejadas para o início das aulas, onde são apresentados aos alunos o ambiente de simulação e também é sugerido alguns trabalhos, para que os alunos tenham tempo de propor seus projetos para a disciplina. Então, nas próximas 3 atividades, são marcadas as datas de entrega e apresentação dos trabalhos. Finalmente, a última atividade listada, de monitoria, deve ocupar praticamente todos os meses com aulas da disciplina, sendo que essa e a atividade que faz o acontecer mais facilmente por parte dos alunos, que devem ser encorajados a procurar a monitoria sempre que estejam com dúvidas.

De um modo geral, a metodologia cumpriu com o esperado. Os alunos projetaram e desenvolveram uma arquitetura de processador *multi-core*, simularam esta arquitetura em um ambiente completo com sistema operacional, rede e programas reais. Além disto, os alunos, ao final da disciplina conseguiram desenvolver e avaliar um programa paralelo para a arquitetura proposta.

3.8. Conclusões

Os processadores *multi-core* e *many-core*, estão se tornando realidade, e representam um novo paradigma em arquiteturas de computadores, uma vez que esses diversos núcleos de processamento disponíveis nos atuais e futuros processadores devem tirar proveito do paralelismo no nível de fluxo de instruções.

Nesse panorama, o ensino de arquiteturas paralelas é de grande importância para a formação das futuras gerações de cientistas da computação. Esse ensino de arquiteturas paralelas deve envolver não apenas os núcleos de processamento desses novos processadores, mas também, todos os dispositivos que vão oferecer suporte adequado para o funcionamento desses núcleos de processamento.

Visando assim, o ensino de arquiteturas paralelas, esse capítulo apresentou uma experiência de ensino de novas arquiteturas, auxiliado por um ambiente de simulação completo.

Podemos observar que o simulador Simics, apresentado nesse capítulo, possui diversas características positivas, como controle, flexibilidade, parametrização, e demais funcionalidades que justificam o seu uso em sala de aula, como ferramenta de ensino para disciplinas de arquitetura paralela.

Através dos dois anos de experiência de utilização desse simulador em sala de aula, podemos notar a importância do profundo entendimento do funcionamento da ferramenta e do bom planejamento de atividades, para que os alunos possam tirar proveito de todos os recursos disponíveis em prol do aprendizado.

Entretanto, devemos considerar a adoção de um plano de melhorias contínuas ao longo dos anos, a fim de aprimorar os materiais didáticos e manuais de uso do simulador, cronogramas de atividades e planos de ensino. Dessa maneira, podemos fornecer cada vez mais suporte aos alunos, mudando também o foco para as futuras arquiteturas e pesquisas.

3.9. Agradecimentos

Agradecemos ao CNPq, CAPES e FAPEMIG que contribuíram com o financiamento para o desenvolvimento deste trabalho.

Referências

[1] UNGERER, T.; ROBIC, B.; SILC, J. A survey of processors with explicit multithreading. *ACM Computing Surveys*, v. 35, n. 1, p. 29–63, 2003.

[2] UNGERER, T.; ROBIC, B.; SILC, J. Multithreaded processors. *British Computer Society*, v. 45, n. 3, p. 320–348, 2002.

[3] HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*. Fourth. USA: Elsevier, Inc., 2007.

[4] SMITH, J. E.; SOHI, G. S. The microarchitecture of superscalar processors. *IEEE*, v. 83, n. 12, p. 1609–1624, 1995.

[5] STALLINGS, W. *Computer Organization and Architecture: Designing for Performance*. Fourth. USA: Prentice Hall, 1996.

[6] OLUKOTUN, K. et al. The case for a single-chip multiprocessor. In: *Proceedings ASPLOS: Int. Symp. on Architectural Support for Programming Languages and Operating Systems*. [S.l.]: IEEE, 1996. p. 2–11.

[7] SINHAROY, B. et al. Power5 system microarchitecture. *IBM Journal of Research and Development*, v. 49, n. 4/5, 2005.

[8] BARROSO, L. A. et al. Piranha: a scalable architecture based on single-chip multiprocessing. In: *Proceedings ISCA: Int. Symp. on Computer Architecture*. [S.l.]: IEEE, 2000. p. 282–293.

[9] KONGETIRA, P.; AINGARAN, K.; OLUKOTUN, K. Niagara: a 32-way multithreaded sparc processor. *IEEE Micro*, v. 25, n. 2, p. 21–29, 2005.

[10] KUMAR, R. et al. Heterogeneous chip multiprocessors. *IEEE Computer*, v. 38, n. 11, p. 32–38, 2005.

[11] FREITAS, H. C.; NAVAUX, P. O. A. *Chip Multithreading: Conceitos, Arquiteturas e Tendências*. Dissertação (Trabalho Individual) — Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil, 2006.

[12] SPRACKLEN, L.; ABRAHAM, S. Chip multithreading: Opportunities and challenges. In: *Proceedings HPCA: Int. Symp. on High-Performance Computer Architecture*. [S.l.]: IEEE, 2005. p. 248–252.

[13] ACOSTA, C. et al. A complexity-effective simultaneous multithreading architecture. In: *Proceedings Int. Conf. on Parallel Processing*. [S.l.: s.n.], 2005. p. 157–164. ISSN 0190-3918.

[14] KOUFATY, D.; MARR, D. T. Hyperthreading technology in the netburst microarchitecture. *IEEE Micro*, v. 23, n. 2, p. 56–65, 2003.

[15] GONÇALVEZ, R.; NAVAUX, P. O. A. Improving smt performance scheduling processes. In: *Proceedings Euromicro, Workshop on Parallel, Distributed and Network-based Processing*. [S.l.: s.n.], 2002. p. 327–334.

[16] EGGERS, S. J. et al. Simultaneous multithreading: A platform for next generation processors. *IEEE Micro*, p. 12–19, 1997.

[17] HAMMOND, L. et al. The stanford hydra cmp. *IEEE Micro*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 20, n. 2, p. 71–84, 2000. ISSN 0272-1732.

[18] KUMAR, R. et al. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. In: *Proceedings SIGARCH Comput. Archit. News*. New York, NY, USA: ACM, 2004. v. 32, n. 2, p. 64. ISSN 0163-5964.

[19] WOLF, W. The future of multiprocessor system on chip. In: *Proceedings DAC: Design Automation Conf.* [S.l.: s.n.], 2004.

[20] ROSE, C. A. F. D.; NAVAUX, P. O. A. *Arquiteturas Paralelas*. Porto Alegre RS, Brasil: Editora Sagra Luzzatto, 2003.

[21] CULLER, D. E.; SINGH, J. P. *Parallel Computer Architecture: A Hardware/Software Approach*. San Francisco, USA: Morgan Kaufmann, 1999.

[22] PATTERSON, D. A.; HENNESSY, J. L. *Organização e projeto de computadores*. Second. Rio de Janeiro, Brasil: Editora Campus, 2005.

[23] BJERREGAARD, T.; MAHADEVAN, S. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, v. 38, p. 1–51, 2006.

[24] JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. New York, USA: J. Wiley, 1991.

[25] JOHN, L. K.; EECKHOUT, L. *Performance Evaluation and Benchmarking*. Boca Raton, USA: Taylor & Francis Group, 2006.

[26] MAGNUSSON, P. et al. Simics: A full system simulation platform. *IEEE Computer Micro*, v. 35, n. 2, p. 50–58, Feb 2002. ISSN 0018-9162.

[27] SIMONG. *Inaccurate Simulation*. Simics Forum, 2007. Disponível em: <https://www.simics.net/mwf/topic_show.pl?pid=55029>. Acesso em: agosto 2008. Disponível em: <https://www.simics.net/mwf/topic_show.pl?pid=55029>.

[28] VIRTUTECH SIMICS. *Simics 3.0 – User Guide for Unix*. [s.n.], 2007. Disponível em: <http://www.simics.net>. Acesso em: junho 2008. Disponível em: <http://www.simics.net>.

[29] THOZIYOOR, S. et al. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In: *Proceedings ISCA: Int. Symp. on Computer Architecture*. [S.l.: s.n.], 2008. p. 51–62.

[30] ALAMELDEEN, A. R. et al. Evaluating non-deterministic multi-threaded commercial workloads. *Computer Architecture Evaluation using Comercial Workloads*, 2002.

[31] ALAMELDEEN, A. R.; WOOD, D. A. Variability in architectural simulations of multi-threaded workloads. In: *Proceedings HPCA: Int. Symp. on High-Performance Computer Architecture*. [S.l.: s.n.], 2003. p. 7–18.

Capítulo 4

Modelos para Programação em Arquitetura Multi-core

Alfredo Goldman (USP - gold@ime.usp.br)
Alberto Hideki Ueda (USP - alberto.ueda@usp.br
Camila Mari Matsubara (USP - camila.matsubara@usp.br)
João Paulo dos Santos Mota (USP - joao.mota@usp.br)

Abstract

Given the ease of reaching the physical limits of performance computing, parallel tasks on multi-core processors is a natural way for technological development. In this sense, learning new paradigms becomes essential. In this chapter, we guide this study based on Programming Models, tools that intermediate building software and its implementation in parallel. A programming model is an interface that provides mechanisms for control over parallelization system. We describe the models classified according to the criteria of level of abstraction. In addition, we present the advantages and disadvantages of each class and some implementation examples. Finally, we discuss the influence of the approach in teaching parallel programming, as well as paths for which parallel computing probably will follow.

Resumo

Diante da facilidade para se alcançar os limites físicos de desempenho computacional, a paralelização de tarefas em múltiplos núcleos de processamento é um caminho natural para a evolução tecnológica. Nesse sentido, o aprendizado de novos paradigmas passa a ser essencial. Neste capítulo, orientamos tal estudo com base em Modelos de Programação, recursos intermediários entre a construção de um software e sua execução em paralelo. Um modelo de programação é uma interface que disponibiliza mecanismos de controle sobre a paralelização do sistema. Descrevemos os modelos classificados conforme o critério de nível de abstração. Além disso, apresentamos as vantagens e desvantagens de cada classe e alguns exemplos de implementação. Por fim, discutimos a influência da abordagem no ensino da programação paralela, assim como os possíveis caminhos para os quais a computação paralela tende seguir.

4.1. Introdução

Este capítulo apresenta os tópicos iniciais a serem abordados no início do processo de ensino de programação paralela. Apresenta também as principais motivações para o aprofundamento nos estudos desta área e a importância dos modelos no suporte ao avanço do paralelismo.

Partindo de conceitos básicos tais como o que é processamento paralelo, veremos rapidamente algumas vantagens deste sobre o processamento sequencial, possíveis desafios dessa abordagem na resolução de problemas, os tipos de arquitetura possíveis e, por fim, modelos de programação aplicados em ambiente paralelos, que serão o assunto principal deste capítulo.

Podemos dizer simplificadamente que a computação paralela se baseia no conceito de divisão-e-conquista: quando se tem uma tarefa demasiada complexa ou que exija grande capacidade de processamento, uma abordagem interessante é dividir em tarefas menores, de preferência independentes, e a partir daí construir a solução para o problema [23].

Pode-se fazer uma analogia interessante com uma montadora de carros: as peças, faróis, painel, bateria, entre outros, são confeccionadas ao mesmo tempo em diferentes lugares para posteriormente formarem um automóvel. Desta forma a produção é otimizada, poupando tempo de fabricação, equipamentos e, consequentemente, poupando recursos financeiros. Este é princípio básico da computação paralela.

Embora a tecnologia dos processadores se desenvolva em um ritmo surpreendente (dobrando sua capacidade a cada 18 meses, segundo revisões recentes da Lei de Moore[1] [18]), estamos em uma época em que os computadores sequenciais aproximam-se dos limites físicos. A computação paralela se torna deste modo uma solução viável para aumentar a eficiência no processamento e o desempenho dos computadores atuais. Nesse contexto, o **paralelismo** surge com a intenção de superar três limites importantes [22]: *Power Wall*, a capacidade de dissipação do calor; *Memory Wall*, a eficiência na comunicação com a memória e *Frequency Wall*, a frequência do cristal oscilador. Estes limites são descritos com mais detalhes na seção 4.2.

Portanto, a computação paralela é um objeto de grande estudo tanto por parte da área acadêmica como por parte das grandes corporações comerciais. Decidir quais tarefas podem ser executadas independentemente, maneiras ótimas de divisão dessas tarefas, aumento da velocidade de comunicação entre os processadores, desenvolvimento de linguagens com suporte ao paralelismo, estes são apenas alguns dos aspectos relevantes quando se visa a construção de sistemas paralelos eficientes.

Baseado nesse contexto, este trabalho aborda os modelos de computação paralela. Estes modelos são espécies de interface entre o baixo nível, ou seja, a implementação do *hardware*, e o alto nível. Tais modelos permitem ao programador abstrair de detalhes da implementação de baixo nível e se concentrar no desenvolvimento do *software*.

Utilizando esta abstração temos que, em geral, computadores paralelos possuem uma arquitetura básica, constituída pelos seguintes componentes: processadores (que são

[1] http://www.intel.com/technology/mooreslaw

basicamente os mesmo de computadores sequenciais), memória e uma rede de comunicação. Além disso, podemos diferenciar os computadores, entre outros aspectos, pelo modo como lidam com a organização da memória: *distribuída* (cada processador possui a sua própria memória) ou *compartilhada*. É importante frisar que estas possíveis organizações de um computador paralelo têm papel importante na elaboração de um modelo de computação paralela.

O capítulo está organizado na seguinte estrutura: Na seção 4.2 enumeramos fatos que favorecem ou desfavorecem a computação paralela, a fim de entender melhor o seu contexto e o impacto de inovações tecnológicas que ela representa. Na seção 4.3 é definido o conceito de modelo de programação paralela. As seções de 4.4 até 4.9 definem a hierarquia de classes de modelos: a estrutura escolhida para a organização destes diferentes modelos. Na seção 4.10 estudamos importantes modelos, principalmente para a última década da área computacional, que não foram incluídos na classificação[2]. Por fim, na seção 4.11 fazemos alguns comentários sobre a influência da abordagem escolhida no ensino da programação paralela, assim como os possíveis caminhos para os quais a computação paralela tende seguir.

4.2. Vantagens e Desvantagens do Paralelismo

Discutir vantagens e desvantagens é um eficiente modo de estimular o interesse no assunto e fixar o aprendizado dos conceitos teóricos sendo aplicados na prática. Nesta seção sugerimos explorar os principais motivos pelo qual o paralelismo têm conquistado cada vez mais destaque [24].

No mundo real, as ações e acontecimentos ocorrem de forma paralela. Certos aspectos da cognição humana são paralelos, como fazer duas coisas ao mesmo tempo, por exemplo. Como a programação é uma forma de expressão das computações do mundo real, é natural que estas possam ser elaboradas em máquinas paralelas e que se tenha maior interesse neste tipo de computação.

No entanto, o que ocorre é que a maior parte dos programas são escritos de forma sequencial e para que isso ocorra, acaba-se impondo uma ordem aos acontecimentos que na verdade são independentes. Uma consequência dessa ordenação de eventos é a combinação de partes do código que exigem uma ordem de ocorrência e partes que são independentes. Essa combinação dificulta o discernimento das relações dos trechos do código, atrapalhando o entendimento do programa e dificultando o aprendizado.

Além do entendimento do código, a ordenação artificial das ações também interfere na sua compilação, sendo mais difícil para o compilador perceber quais partes do código são passíveis de troca de ordem de execução e quais não são.

Aliada à conveniência do paralelismo do mundo real, a paralelização das computações também traz a vantagem do maior desempenho. Apesar da obtenção deste desempenho não ser trivial, existem áreas que exigem essa melhora, pois empregam grandes processamentos e as melhores técnicas sequenciais ainda não são suficientes, como é o caso da previsão do tempo [5], por exemplo.

[2]Usamos como base a classificação inicialmente proposta por Skillicorn[24]

Porém, a dificuldade de aumento do desempenho não está apenas na implementação do código, mas também na capacidade física da máquina que confronta fundamentos físicos, como o número de transistores dentro de um processador. Um processador único possui seus limites que já podem estar próximos da capacidade atual. Além disso, o potencial do processador pode ser limitado por fatores econômicos, como por exemplo, o desenvolvimento de processadores de silício e arsenieto de gálio que melhoram o desempenho, mas tem custo elevado. Os três principais limites [22] são:

- *Power Wall*: O incremento no desempenho dos processadores está limitado pela capacidade de dissipação de calor e energia;
- *Memory Wall*: O desempenho de programas está limitado pela eficiência na atividade de transferir dados da memória principal para o processador e vice-versa;
- *Frequency Wall*: A capacidade de processamento do processador depende da frequência do cristal oscilador, dada em ciclos (*clocks*). Quanto mais ciclos por segundo, mais operações são executadas em um mesmo tempo. A velocidade máxima possível é a da luz[3], cujo limite está próximo de ser alcançado.

Processadores ópticos surgem como possibilidade para solução do problema de desempenho, mas é uma solução que só deve estar disponível em algumas décadas. Assim, o emprego do paralelismo interno (processadores superescalares), que permite a execução de mais de uma instrução simultaneamente, vem ajudando a solucionar o problema e, mesmo que a velocidade dos processadores continue a aumentar no ritmo que apresenta atualmente, o paralelismo interno pode ainda ser uma alternativa economicamente mais interessante para algumas aplicações.

Novas tecnologias têm atrelado ao seu preço o custo de projeto e fabricação que elevam o seu valor e também diminuem consideravelmente o preço de gerações anteriores. Como a melhora no desempenho de geração para geração tem sido de aproximadamente uma ordem decimal, a união de processadores de gerações anteriores, também dependendo do custo de conexão, pode obter velocidades de processamento próximas dos de última geração com custo reduzido.

Portanto, os altos preços de processadores de última geração, a diminuição do preço de gerações anteriores e a ordem de magnitude de melhora de desempenho entre as gerações favorecem o projeto inteligente e ajudam na disseminação do uso de redes de trabalho com um conjunto de inúmeros computadores de baixo custo.

Entretanto, também existem fatores que complicam a paralelização computacional. Atividades que são executadas simultaneamente adicionam detalhes que não aparecem na computação sequencial. Um exemplo é a dependência de um mesmo recurso por várias atividades que são executadas em paralelo. É necessário maior atenção e cuidado ao lidar com recursos compartilhados neste caso.

Além deste tipo de complicações, o fato de a teoria para computação paralela ter iniciado seus estudos após a criação de sua tecnologia faz com que esta ainda não possua uma base sólida e ainda restem dúvidas quanto a representações abstratas, lógica de

[3]Na verdade, o limite é ainda menor no caso da eletricidade em fios de cobre, por exemplo

raciocínio e algoritmos eficazes, dificultando a obtenção de resultados reais ótimos. A área ainda não foi totalmente explorada e exige mais estudos para que atinja sua maturidade.

Para o alcance da maturidade no estudo do paralelismo ainda é necessário o entendimento do equilíbrio entre as partes do computador paralelo e sua influência no desempenho. Fatores como a velocidade do processador, a comunicação, e o desempenho da hierarquia de memória afetam o alcance de resultados que se consideram atingíveis na teoria e que ainda não foram alcançados.

Além do equilíbrio e desempenho, estes fatores também influenciam na portabilidade do código. Isso porque, entre computadores monoprocessados, um mesmo programa não obtém desempenho muito variável de arquitetura para arquitetura, porém, em máquinas paralelas pode se obter diferenças de uma ordem de magnitude. Essa diferença de desempenho é dependente de estruturas de conexão e do custo da comunicação. Por este motivo a portabilidade é afetada e, para que ela seja possível, pode ser necessária a reconstrução do *software*, desestimulando o paralelismo.

Porém, o crescimento de computadores pessoais multiprocessados no mercado estimula o desenvolvimento de programas paralelizados, uma vez que a maioria dos aplicativos atuais, planejados essencialmente para um único processador, não utilizam todo o potencial de um processador multi-core.

4.3. Visão Geral sobre os Modelos

Um modelo de computação paralela é uma interface responsável por separar propriedades de alto nível das de baixo nível. Em outras palavras, um modelo é uma máquina de abstração que fornece operações para a programação paralela. É projetado para separar o que diz respeito a desenvolvimento de *software* do que diz respeito a execução paralela efetiva; e estabelece ao desenvolvedor tanto abstração quanto estabilidade.

Abstração aparece porque as operações que o modelo fornece são de mais alto nível, simplificando a estrutura do *software* e reduzindo as dificuldades na sua construção. Estabilidade aparece porque a construção do *software* pode supor uma interface padrão, independente dos detalhes da arquitetura do computador paralelo.

Além disso, um modelo de programação reproduz restrições de máquinas concretas existentes; tenta ter máxima aplicabilidade em máquinas existentes e também nas futuras; faz uma estimativa do desempenho e do custo do programa [11].

Como modelos são considerados máquinas abstratas, existem diferentes níveis de abstração. Para a apresentação dos modelos de programação em arquitetura multi-core, sugerimos aqui uma classificação de acordo com esse nível de abstração e de detalhamento, baseada em [24], e a descrição dos mesmos em ordem decrescente.

É importante ressaltar que a maioria dos modelos não foi criada com a ambição de funcionar bem para todos os casos. Para evitar que o estudo sobre modelos se torne demasiadamente complexo e exaustivo, esta classificação não aborda todos os modelos que já existiram, e sim aqueles que trouxeram alguma inovação importante e suas principais características.

Em cada classe, há diferentes níveis de grau de controle sobre a estrutura e a comunicação, o que permite que se possa ter diferentes visões sobre o paralelismo e introduz ao programador as opções disponíveis de controle e decisões sobre a paralelização bem como as distintas formas de raciocínio utilizadas para cada tipo de modelo. Assim, tem-se diferentes abordagens do ensino do raciocínio paralelo e são apresentadas algumas linguagens utilizadas para a implementação de programas paralelizados. A seguir descrevemos o conteúdo de cada classe de modelos.

4.4. Nada Explícito

Quanto mais um modelo de computação paralela afastar o programador das decisões sobre o paralelismo, melhor ele deve ser considerado. Entretanto modelos como este são muito abstratos e difíceis de implementar, devido a grande quantidade de inferências que são necessárias para realizar a paralelização de forma totalmente automatizada.

A automatização deve estar em todas as fases da paralelização: dividir o programa em partes adequadas, mapear essas partes nos processadores, escaloná-las e sincronizá-las. Porém, apesar do muito trabalho investido, essa abordagem se mostrou ineficaz devido a dificuldade em se determinar de antemão se é essencial paralelizar um determinado aspecto do programa. Isto só é possível se desde o início do projeto for escolhido um modelo voltado para esta questão, porém isso é muito abstrato e custoso. As abordagens propostas mantendo alto nível de abstração são variadas e mostradas nesta seção.

4.4.1. Programação funcional de ordem superior

Programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, ao invés de enfatizar mudanças no estado do programa. Dessa maneira, o resultado de um problema computacional pode ser entendido como uma solução de um sistema gerado a partir dessas funções.

A técnica utilizada para computar o valor de uma função é chamada *graph reduction* [21]. Nela as funções são representadas como árvores e a técnica consiste em selecionar subestruturas do grafo, reduzi-las a formas mais simples e recolocá-las no grafo. É fácil perceber que essa abordagem é paralelizável, afinal duas partes do grafo com intersecção vazia podem ser valoradas separadamente e simultaneamente. Então vários processadores podem procurar por partes redutíveis independentemente.

Infelizmente essa ideia não costuma funcionar. Pois apenas as reduções que contribuem para o resultado final deveriam ser realizadas, já que calcular outras seria, além de um desperdício de recursos, uma alteração na lógica do programa. Fica muito difícil determinar de forma eficiente que partes independentes do programa necessitam ser computadas de fato para se chegar ao resultado final. Isso só é possível a partir de uma análise sofisticada do programa como um todo. Logo, esse modelo não é capaz garantir bom desempenho ou ter seu custo estimado. Por essa e outras dificuldades a *parallel graph reduction* teve um sucesso limitado para computadores de memória compartilhada. Para os computadores de memória distribuída ainda não se sabe qual é a efetividade dessa abordagem.

4.4.2. Interleaving

Uma segunda abordagem possível é a *Interleaving* (entrelaçamento), que descreve um método para organizar dados de maneira não contígua para aumentar o desempenho do programa. É uma técnica onde os bits de um bloco de dados são entrelaçados com o uso de uma matriz. O entrelaçamento consiste simplesmente da mudança na ordem dos bits. Esta abordagem deriva de ideias de multiprogramação em sistemas operacionais e é usado principalmente em sistemas de transmissão e comunicação de dados. Se o objeto a ser computado pode ser dividido em um conjunto de sub-computações que comutam então existe uma liberdade considerável e conveniente para o *Interleaving*.

Cada parte do código é associada a uma expressão booleana. Na computação do programa todas as expressões são avaliadas e aquelas que aparecem como verdadeiras terão seus sub-programas executados. Esse processo é então repetido.

Como a abordagem anterior, modelos de *Interleaving* são simples e abstratos, porém é difícil garantir bom desempenho e impossível avaliar seu custo.

4.4.3. Linguagens de lógica paralela implícita

Linguagens de lógica implícita exploram o fato de que o processo de resolução de pesquisa lógica contém muitas atividades que podem ser feitas em paralelo. Os principais tipos de paralelismo inerentes de programas lógicos são aqueles causados pelo OU e pelo E. Assim, as duas sentenças envolvidas no OU ou no E podem ser avaliadas simultaneamente.

Linguagens de lógica paralela implícita permitem decomposição automática da árvore de execução em uma rede de processos paralelos. Isso é possível pelo suporte da linguagem à análise sintática tanto na compilação quanto na execução.

Como as outras abordagens já discutidas, essa abordagem é bastante abstrata e não pode garantir bom desempenho, apesar de às vezes consegui-lo. Avaliações de custo não podem ser feitas porque *linguagens de lógica implícita* são muito dinâmicas, isto é, interpretam apenas em tempo de execução muitos comportamentos que outras linguagens interpretam durante a compilação.

4.4.4. Skeletons algorítmicos e homomóficos

Esta classe de modelos propõem alto nível de abstração. A programação abstrata é baseada em unidades ou componentes fundamentais cujas implementações estão pré-definidas. Em outras palavras, os programas são construídos conectando-se blocos prontos (*building blocks*). Esses blocos mantém o nível de abstração alto porque são unidades fundamentais com as quais o programador trabalha, e podem ocultar uma quantidade arbitrária de complexidade. No contexto de programação paralela, estes blocos são chamados *skeletons*.

Skeletons algorítmicos são aqueles que encapsulam as primitivas de controle e comunicação da aplicação [23]. Cada *skeleton* corresponde a uma padronização de algum algoritmo ou fragmento de algoritmo, e pode-se compor um conjunto de *skeletons* sequencialmente. O programador escolhe quais blocos deseja usar, o compilador escolhe como cada algoritmo será implementado e como será a paralelização dentro de cada *skeleton* e entre eles.

A abordagem de *skeletons* algorítmicos é simples e abstrata. Entretanto, como os programas precisam ser expressos como composição dos *skeletons* dados, a expressividade de linguagens de programação abstrata baseadas em *skeletons* algorítmicos é algo que ainda está em aberto. Por outro lado é possível garantir bom desempenho com os *skeletons* se eles forem bem escolhidos, o que também leva a uma possível avaliação de custo.

Já o modelo *skeleton homomórfico* é baseado em estruturas de dados. Os *skeletons* nesse modelo são organizados em conjuntos, um para cada estrutura de dados específica: um para listas, um para árvores, etc. Todos os *homomorfismos* em uma estrutura de dados podem ser implementados em padrão computacional recursivo e paralelizável. A comunicação necessária por estes tipos de *skeletons* é dedutível a partir de cada estrutura de dados. Porém é muito comum que não sejam muitos os requisitos de comunicação.

A abordagem de *skeleton homomórfico* também é simples e abstrata, e o método de construção dos *homomorfismos* das estruturas de dados naturalmente gera um ambiente rico para as transformações equacionais. É possível utilizar esse modelo com bom desempenho, além de permitir uma avaliação de custo.

4.4.5. Linguagens de processamento celular

Linguagens de processamento celular são baseadas no modelo de execução de um *autômato celular*. Um autômato celular consiste de uma distribuição *n*-dimensional de células, onde cada célula está conectada a um conjunto finito de células ditas adjacentes. Uma célula tem um estado escolhido de um alfabeto finito. O estado de um autômato celular é completamente definido pelos valores das variáveis em cada célula. As células têm seus valores atualizados através de uma função de transição que tem como parâmetro o estado da célula atual e de um conjunto de células próximas a ele (a uma distância fixada).

Nesse ambiente é possível que as funções de transição sejam aplicadas simultaneamente em partes distintas do autômato possibilitando a paralelização. Exemplos de linguagens com esse tipo de abordagem são: Cellang, CARPED, CDL e CEPROL. Essa abordagem permite bom desempenho tanto em sistemas de memória distribuída quanto compartilhada.

4.5. Paralelismo Explícito

A segunda maior classe de modelos é essa em que o paralelismo é explícito em programas abstratos mas o desenvolvedor não precisa explicitar como os cálculos computacionais serão divididos em partes nem como essas partes serão mapeadas entre os processadores ou irão se comunicar e sincronizar.

4.5.1. Dataflows

No modelo *Dataflow* tudo o que há para ser computado é representado como operações, geralmente pequenas, e que possuem entradas e resultados explícitos. A dependência entre os dados do programa determina a ordem de execução dessas operações. Operações que não possuem dependências são computadas concorrentemente.

As operações de um programa são conectadas por caminhos (que expressam as

dependências de dados) pelos quais os dados fluem - daí o nome do modelo. O compilador pode dividir o grafo que representa essa computação de qualquer forma. Além disso, as operações são executadas em uma ordem que depende apenas dos processadores que estão livres em um dado momento - já que a dependência já está descrita pelos caminhos do grafo. Por isso, a decomposição tem pouco efeito sobre o desempenho, e pode ser feita automaticamente. Até mapeamentos aleatórios de operações em processadores rodam bem na maioria dos sistemas de *Dataflow*.

Linguagens de *Dataflow* são abstratas e simples, porém não possuem uma metodologia natural de desenvolvimento de *software*. Essas linguagens podem garantir desempenho: de fato, a linguagem Sisal compete em eficiência com Fortran, em arquiteturas com memória compartilhada. No entanto, o desempenho em arquiteturas com memória distribuída ainda não é competitiva. Como muito agendamento é feito dinamicamente em tempo de execução medidas de custos não são possíveis.

4.5.2. Linguagens lógicas explícitas

Também são chamadas de linguagens lógicas concorrentes, e podem ser vistas como uma nova interpretação de cláusulas Horn (cláusulas com no máximo 1 literal positivo). Isso significa que uma cláusula com objetivo atômico $\leftarrow C$ pode ser visto como um processo; um objetivo conjuntivo $\leftarrow C_1, ..., C_n$ pode ser visto como uma rede de processos; e uma variável lógica usada entre dois subobjetivos distintos pode ser vista como um canal de comunicação entre dois processos. Exemplos de linguagens dessa classe são PARLOG [4], Delta-Prolog, Concurrent Prolog. O paralelismo é alcançado pelo enriquecimento de uma linguagem lógica (como o Prolog) com alguns mecanismos para a anotação de programas.

Uma outra linguagem onde o paralelismo deve ser explicitado pelo desenvolvedor é o Multilisp. Esta linguagem é uma extensão de Lisp onde oportunidades para o paralelismo são criadas usando a anotação *future*. Nela, há uma correspondência direta entre *threads* e *futures*. A expressão *future X* representa que o valor X está parcialmente calculado, e cria um processo para computar X, possibilitando paralelização entre o processo que deve computar aquele valor e o processo que deve usar aquele valor. Quando o valor de X é computado, o valor substitui o *future*.

4.5.3. Estrutura estática

O modelo de estrutura estática baseia-se no paralelismo de dados, isto é, na aplicação de operações monolíticas em objetos de uma estrutura de dados. Inicialmente, parece um modelo de programação de expressividade limitada, mas é uma forma poderosa de descrever muitos algoritmos interessantes. Os algoritmos são analisados no sentido de encontrar situações em que uma mesma operação é aplicada repetidamente em diferentes dados sem que essas operações interajam. Por exemplo, tome um vetor como a estrutura de dados em questão e uma função que mapeia cada elemento do vetor em um novo elemento que depende exclusivamente do valor inicial. Nesse caso o mapeamento de cada elemento é independente, o que corresponde a uma operação monolítica sobre uma estrutura de dados.

Por exemplo, considere a linguagem Fortran com a adição de um laço *ForAll*, onde

as interações são independentes e podem ser executadas concorrentemente. O seguinte trecho de código é paralelizável [24]:

```
ForAll (I = 1:N, J = 1:M)
    A(I, J) = I * B(J)
```

É preciso ter cuidado para que os laços não usem o mesmo endereço entre expressões diferentes. O compilador nem sempre pode verificar isto, portanto é tarefa para o programador fazer esta verificação.

Muitos dialetos de Fortran, como o Fortran-D ou o HPF (*High Performance Fortran*) utilizam esse tipo de paralelismo e adicionam mais paralelismo direto, incluindo construções para especificar como as estruturas serão alocadas aos processadores e operações de redução.

Neste tipo de modelos, existe um desempenho garantido e ainda é possível medir os custos.

4.6. Decomposição Explícita

Nesta classe de modelos o paralelismo e a decomposição são explícitos, porém o mapeamento, comunicação e sincronismo são implícitos. É importante deixar claro que tais modelos precisam de decisões quanto à quebra do trabalho em pedaços, porém deixam ao cargo do desenvolvedor do *software* as implicações destas decisões. Ou seja, modelos deste tipo exigem programas abstratos que especifiquem os pedaços em que serão divididos, mas a disposição destes segmentos nos processadores e a maneira com que eles vão se comunicar não precisa ser descrita explicitamente.

Os principais modelos representantes desta classe são BSP (*Bulk Synchronous Parallelism*) e LogP [16]. A apresentação destes 2 modelos se mostra suficiente para a compreensão do conceito e das características da Decomposição Explícita.

4.6.1. BSP

Bulk Synchronous Parallelism é um modelo ideal para aplicações cuja comunicação é baseada em troca de mensagens. As propriedades da rede de interconexão deste modelo são capturadas através de alguns parâmetros da arquitetura. Uma máquina abstrata de BSP consiste de uma coleção de processadores abstratos, cada um com memória local, conectados por uma rede de interconexão (arquitetura de memória distribuída), cujas únicas propriedades de interesse são:

- o tempo para fazer uma sincronização no barramento, l;
- o grau de velocidade com que dados constantes continuamente e aleatoriamente endereçados podem ser entregues, g.

Esses parâmetros de BSP são determinados experimentalmente para cada computador paralelo.

Um programa (abstrato) de BSP consiste de *p threads* e é dividido em *supersteps* [11]. Cada *superstep* consiste de: uma computação em cada processador, usando apenas valores mantidos localmente; uma transmissão de mensagem global de cada processador para qualquer conjunto de outros processadores; e um barramento de sincronização. Ao final de um *superstep*, os resultados das comunicações globais tornam-se visíveis no ambiente local de cada processador.

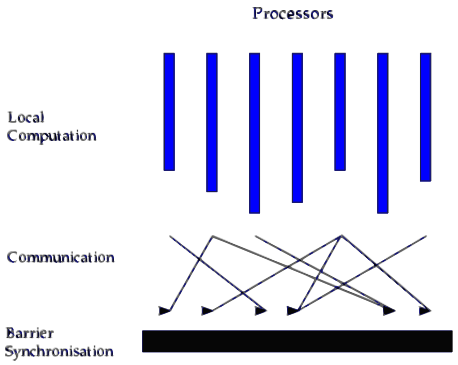


Figura 4.1. *superstep* de um programa BSP - figuras são ferramentas poderosas no ensinamento de conceitos. Fonte: [24].

Intuitivamente, o tempo de cada *superstep* é da ordem de: o tempo dos cálculos locais, mais o tempo de todas as comunicações, mais o tempo da sincronização no barramento. Formalmente temos: se a computação local máxima em um passo demorar tempo w e o número máximo de valores enviados ou recebidos por qualquer processador é h, então o tempo total t para um *superstep* é dado por

$$t = w + hg + l$$

onde g e l são os parâmetros de rede. Logo, é fácil determinar o custo de um programa, esse tempo estimado depende da aleatoriedade do mapeamentos das *threads* e da estimativa do tempo limite para a comunicação.

Portanto, programas BSP devem ser decompostos em *threads*, mas o mapeamento das *threads* é feito automaticamente. A comunicação é derivada do mapeamento das *threads*, e a sincronização depende do programa como um todo. O modelo é simples e convenientemente abstrato, mas necessita de uma metodologia de construção de *software*. A medição de custos apresenta o custo real de um programa em qualquer tipo de arquitetura para o qual g e l são conhecidos.

Após a conclusão da apresentação do modelo, sugerimos uma descrição do estado do modelo nos dias atuais como uma forma de motivação do estudo, associado a um exemplo de aplicação.

A versão mais atual de implementação de BSP usa um biblioteca SPMD (*Single-Program Multiple-Data*), que pode ser nas linguagens C ou Fortran. A biblioteca oferece operações para colocar dados na memória local a partir de um processo remoto (*put*), para acessar os dados de um processo remoto (*get*), e para sincronizar. A seguir, um exemplo de um pequeno programa para computar somas de prefixos [24].

```
1  int prefixsums(int x) {
2      int i, left, right;
3      bsp_pushregister(&left, sizeof(int));
4      bsp_sync();
5      right = x;
6      for (i = 1; i < bsp_nprocs(); i*=2) {
7          if (bsp_pid() + i < bsp_nprocs())
8              bsp_put(bsp_pid() + i, &right, &left, 0, sizeof(int));
9          bsp_sync();
10         if (bsp_pid() >= i)
11             right = left + right;
12      }
13      bsp_popregister(&left);
14      return right;
15  }
```

As chamadas *bsp_pushregister* e *bsp_popregister* são necessárias para que cada processo possa acessar pelo nome variáveis armazenadas em um *heap* ou pilha em processos remotos.

4.6.2. LogP

Outro tipo de abordagem é o *LogP* – mais detalhes sobre esse modelo em [16] –, que usa *threads* com contextos locais, atualizadas por meio de comunicação global. Entretanto, LogP não tem um barramento de sincronização de ponta a ponta, isto é, que alcança todas as threads. O LogP propõe ser um modelo abstrato capaz de capturar a realidade tecnológica da computação paralela. Este modelo de computação paralela usa quatro parâmetros(L, o, g e P – daí o nome do modelo): latência L, *overhead* o, largura da faixa de comunicação (*gap*) g, e o número de processadores P [11]. Um conjunto de programas foi projetado com o modelo LogP e implementados na máquina paralela CM-5 para avaliar a utilidade do modelo. Porém, foi comprovado que o modelo LogP não é mais poderoso que BSP.

O CM-5 (*Connection Machine 5*), é uma máquina da *Thinking Machines Corporation*, de processamento massivamente paralelo, que possui memória local com endereçamento local, a capacidade de difundir e sincronizar instruções, sincronização entre os processadores e a existência de *hardware* especial para suporte ao paralelismo de dados. Tem 16384 processadores, ocupa um espaço de 30 metros quadrados e atinge desempenho de mais de 1 Teraflops (10^{12} operações de ponto flutuante por segundo).

4.7. Mapeamento Explícito

Modelos deste tipo necessitam de comandos para especificar como o código é decomposto em partes e para quais dispositivos estas partes são delegadas, no entanto eles fornecem certa abstração quanto à comunicação e o sincronismo entre as partes. Aqui, o desenvolvedor do *software* deve não só dividir o trabalho em pedaços, mas também deve considerar o melhor lugar para colocar estes pedaços em cada processador. Como a localidade tem efeito no desempenho da comunicação, torna-se inevitável que a rede de interconexão entre os processadores seja bem clara. Por isso, todos os modelos desta classe objetivam reduzir a responsabilidade do desenvolvedor de se preocupar com os cálculos de comuni-

cações.

Para facilitar a apresentação desta categoria de modelos de programação, esta seção está dividida em linguagens de diferentes tipos: de coordenação, de comunicação, funcionais e de coordenação com contexto. Em cada tipo descrevemos as principais características do modelos e apresentamos pequenos exemplos.

4.7.1. Linguagens de coordenação

Linguagens de coordenação simplificam a comunicação, separando os aspectos computacionais dos programas de seus aspectos comunicativos e estabelecendo um padrão próprio para comunicação.

É importante ressaltar que, apesar de classificada em Mapeamento Explícito, a principal preocupação desse tipo de linguagem é facilitar a comunicação, tornando-a mais abstrata para o desenvolvedor – ao invés de se preocupar com técnicas para explicitar o mapeamento. Isso é justificado pelo fato de que a comunicação é a etapa seguinte ao mapeamento.

Começamos a descrição deste tipo de modelo com um exemplo: o modelo Linda [24], que substitui a comunicação ponto-a-ponto por um grande repositório compartilhado pelos processos, e de onde são recuperados associativamente. Este repositório compartilhado é chamado de Espaço de Tuplas [4]. O modelo Linda contém três operações de comunicação:

- IN: retira uma tupla do Espaço de Tuplas, baseado em valores de alguns de seus campos, devolvendo nos parâmetros de quem a requisitou os valores da tupla removida;
- RD (*read*): faz o mesmo que o anterior exceto pelo fato que lê a tupla do Espaço de Tuplas antes de removê-la;
- OUT: coloca uma tupla no repositório.

Por exemplo, a operação

$$rd(\text{“}Brasil\text{”}, ?X, \text{“}Portugal\text{”})$$

procura pelas tuplas que possuam três elementos, em que o primeiro é *Brasil*, último é *Portugal* e o segundo é um elemento do mesmo tipo que o X. Por trás destas três operações básicas, Linda utiliza a operação $eval(t)$ que cria implicitamente um novo processo para avaliar a tupla e inseri-la no Espaço de Tuplas.

As operações de Linda separam as partes de recebimento e envio de informação em uma comunicação - a *thread* de envio não lida com a *thread* de recebimento, nem mesmo sabe se esta existe. Este modelo requer do desenvolvedor o gerenciamento de *threads*, mas reduz a carga imposta pela coordenação da comunicação. Infelizmente, um Espaço de Tuplas não está necessariamente implementado com garantia de desempenho, portanto tal modelo não possibilita medições de custo.

Outro ponto importante é a metodologia de desenvolvimento de *software*. Para tratar disto foi implementado um ambiente de programação de alto nível, chamado *Linda Program Builder* (LPB), para dar suporte ao projeto e desenvolvimento de programas em Linda. Este ambiente guia o usuário para o projeto, codificação e monitoramento de programas e a execução do *software* Linda. Uma descrição completa do LPB é encontrado em [1].

4.7.2. Linguagens de comunicação sem mensagens

Linguagens de comunicação sem mensagens reduzem a carga (*overhead*) de gerenciamento de comunicação simulando esta comunicação de modo que se ajuste mais naturalmente às *threads*. Por exemplo, ALMS [24] lida com a troca de mensagens como se os canais de comunicação fossem mapeados na memória. Referir-se para certas variáveis em diferentes *threads* funciona como transferir uma mensagem de uma para as outras. PCN (*Program Composition Notation*) e *Compositional C++* também ocultam a comunicação por meio de variáveis de uso único. Uma requisição de leitura de uma dessas variáveis bloqueia a *thread* se seu valor ainda não foi mapeado ou atualizado por outra *thread*. Esta abordagem é similar ao uso de bits cheios/vazios em variáveis, uma ideia antiga que volta a ser importante em arquiteturas *multi-threads*.

Em particular, na linguagem PCN, os processos que compartilham uma variável de definição conseguem, por meio desta, se comunicar uns com os outros. Por exemplo, na composição paralela:

$$\{||produtor(X), consumidor(X)\}$$

ambos os processos (produtor e consumidor) podem usar a variável X para se comunicar entre si. Em PCN, o mapeamento dos processos para os devidos núcleos é definido pelo programador ou pelas anotações em suas funções ou ainda por um mapeamento de topologias virtuais para físicas. A extensão lógica do mapeamento de comunicação para a memória é a Memória Compartilhada Virtual, em que a abstração fornecida para o programa é de um único espaço de endereçamento compartilhado, sem a preocupação com a real alocação de memória. Isto requer referências remotas de memória ou para ser compiladas em mensagens ou para ser ativadas por mensagens em tempo de execução. Até aqui, os resultados não sugerem que esta abordagem é escalável.

4.7.3. Linguagens funcionais com anotação

Linguagens funcionais com anotação facilitam o trabalho do compilador permitindo aos programadores fornecer informações extras sobre como particionar a computação em partes e mapeá-las.

Um exemplo deste tipo de linguagem é *Paralf*, uma linguagem funcional baseada em avaliação postergada. Entretanto, *Paralf* permite ao usuário controlar a ordem de avaliação por anotações explícitas. Em *Paralf*, a comunicação e sincronização são implícitas, mas existe uma notação de mapeamento para especificar quais expressões serão avaliadas em cada processador. Uma expressão seguida da anotação $\$onproc$ será analisada no processador identificado como *proc*. Por exemplo, a expressão:

$$(f(x)\$on(\$self+1))*(h(x)\$on(\$self))$$

denota a computação da sub-expressão $f(x)$ em um processador vizinho (em paralelo) com o da execução de $h(x)$.

Uma outra abordagem é o mecanismo de Chamada de Procedimento Remoto (RPC), que é uma extensão da chamada de procedimento tradicional. Um RPC é uma chamada entre dois diferentes processos, o remetente e o destinatário. Quando um processo chama um procedimento remoto de um outro processo, o destinatário executa o código do procedimento e devolve para o remetente os parâmetros de saída. Assim como *rendezvous* (que será apresentado mais a diante), RPC é uma forma de cooperação síncrona, isto é, durante a execução do procedimento, o processo remetente é bloqueado, sendo reativado com o retorno dos parâmetros de saída. A sincronização total de RPC pode limitar o aproveitamento de um alto nível de paralelismo entre os processos que compõem um programa concorrente. Algumas linguagens baseadas neste mecanismo são DP, Cedar e Concurrent CLU.

4.7.4. Linguagens de coordenação com contexto

Linguagens de Coordenação com contexto estendem a ideia de Linda. Um dos pontos desfavoráveis de Linda é que ela fornece um único e global espaço de tuplas e portanto impede um desenvolvimento modular de *software*. Um exemplo é a linguagem *Ease*, que inclui ideias de *Occam*. Programas que adotam *Ease* possuem múltiplos espaços de tuplas, denominados contextos, e podem ser visíveis a apenas algumas *threads*. Como as *threads* que acessam um contexto especifíco são conhecidas, estes contextos assemelham-se a algumas das propriedades de *Occam* - como canais. Mais detalhes sobre *Occam* em [25].

As *threads* leêm e escrevem dados em contextos como se eles fossem espaços de tuplas de Linda. *Ease* possui muitas propriedades do modelo Linda, porém torna mais fácil a implementação com garantia de desempenho. *Ease* também ajuda na decomposição possibilitando a mesma estruturação de processos de *Occam*.

Outra linguagem relacionada é ISETL-Linda, uma extensão do paradigma SETL para computação de conjuntos. Este modelo adiciona os espaços de tuplas de Linda como um tipo de dados e os trata como objetos de primeira classe. Uma outra linguagem semelhante, baseada em Fortran é *Opus*, que possui paralelismo tanto de operações quanto de dados, mas deixa a comunicação ser mediada pelas abstrações de dados compartilhados. Estes são objetos autônomos visíveis para qualquer subconjunto de operações, mas internamente são sequenciais, isto é, somente um único método de cada objeto pode estar ativo por vez. Estes objetos são, na realidade, um tipo de generalização de monitores.

4.7.5. Skeletons de comunicação

São *skeletons* que estendem a ideia de blocos pré-estruturados de construção para a comunicação. Um *skeleton* deste tipo é uma união de níveis computacionais, que consistem em computações locais independentes, com níveis de comunicação, constituídos de padrões fixos em uma topologia abstrata. Estes padrões são coleções de arestas, em que cada aresta funciona como um canal de difusão. A Figura 4.2 mostra um esqueleto usando duas etapas de comunicação, combinando-os com dois padrões diferentes de comunicação. Este modelo é uma junção de ideias de BSP e de *skeletons* algorítmicos, somando-se conceitos

como *Adaptive Routing* e difusão que são suportados pelas novos *designs* de arquitetura.

O modelo é razoavelmente independente da arquitetura, pois os *skeletons* de comunicação podem ser construídos assumindo uma topologia sem conexões especiais como objetivo; os resultados obtidos podem ser usados para construir então implementações para topologias para interconexões mais sofisticadas. Este modelo possibilita garantia de desempenho e apresenta medições de custo.

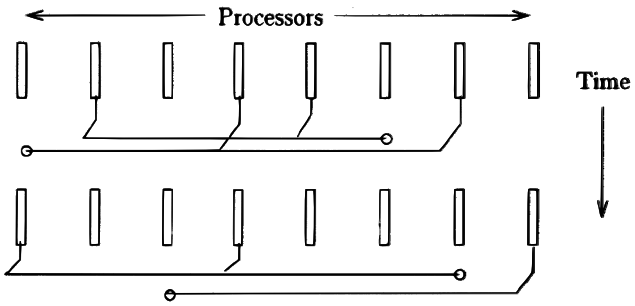


Figura 4.2. Um *skeleton* de comunicação. Fonte: [24].

4.8. Comunicação Explícita

Esta seção apresenta uma abordagem para modelos onde o paralelismo, decomposição, mapeamento e comunicação são explícitos, porém o sincronismo é implícito. Note que neste, o desenvolvedor faz a maior parte das decisões de implementações, e reduzem apenas as responsabilidades de sincronização associadas a comunicação. Geralmente isso é feito com uma semântica assíncrona: mensagens são entregues mas o remetente não depende da entrega, e o recebimento de múltiplas mensagens pode causar travamento.

Para apoiar o desenvolvimento nesta seção, apresentamos os modelos *Actors*, *ActorSpace*, *Concurrent Aggregates*, Orientação a Objetos focado no paralelismo, *Arrays* sistólicos, e algumas linguagens como *Mentat Programming Language* [24].

4.8.1. Actors, ActorSpace e Concurrent Aggregates

O modelo mais importante nesta classe é *Actors*. Foi criado em 1973 por Carl Hewitt, Peter Bishop e Richard Steiger, e publicado no artigo intitulado *A Universal Modular Actor Formalism for Artificial Intelligence* [12].

Nesse modelo mantém-se coleções de objetos (chamados atores), onde cada um deles é tratado como a primitiva universal da computação concorrente. Cada ator tem uma fila de mensagens e executa repetidamente a seguinte sequência:

- lê a próxima mensagem;
- envia mensagens a outros atores cuja identidade ele conhece; e
- toma decisões locais, definindo um novo comportamento que determinará sua resposta para a próxima mensagem.

- Ele também pode criar um número finito de novos atores.

Os nomes de atores são objetos que podem ser passados através de mensagens. Mensagens são entregues assincronamente e desordenadamente. Porém, para *Actors* os itens 'desempenho garantido' e 'medição de custos' não é possível, porque a comunicação total em um programa *Actors* não é limitada. Além disso, como este modelo é altamente distribuído (cada ator toma decisões individualmente), é responsabilidade dos compiladores serializar a execução para obter eficiência em processadores convencionais.

Uma das transformações no compilador mais eficientes é eliminar a criação de alguns tipos de atores e substituir mensagens enviadas a atores no mesmo processador por chamadas de funções. Assim, não é possível determinar o verdadeiro custo de executar um programa *Actor* sem a especificação de como os atores são mapeados para os processadores e *threads*.

Um dos pontos fracos do modelo *Actors* é que cada ator processa sua fila de mensagem sequencialmente e isso pode se tornar um gargalo para o desempenho. Para tratar esse problema, foram propostas duas extensões do modelo *Actors*: *Concurrent Aggregates* e *ActorSpace*.

Concurrent Aggregates (CA) extende o paradigma de Orientação a Objetos para o contexto de computação paralela, incorporando objetos concorrentes, chamada de métodos concorrentemente e agregados (coleção de objetos distribuídos) [8]. Assim, esse modelo se torna bastante conveniente para explorar o paralelismo em computadores massivamente paralelos, no qual não há recursos de serialização desnecessários. CA combina orientação a objetos com concorrência. Essa mistura permite o uso de poderosas ferramentas como a abstração de dados para esconder a complexidade da programação paralela, em particular, o gerenciamento da concorrência, sincronização e distribuição de tarefas. Um 'agregado' em CA é uma coleção homogênea de objetos que são agrupados e referenciados por um único nome. É como se cada agregado fosse um conjunto de atores que possuem características semelhantes. Um agregado é multi-acessível, ou seja, é capaz de receber várias mensagens simultaneamente, diferentemente de outros modelos/linguagens orientada a objetos.

O modelo *ActorSpace* extende o modelo *Actor* para evitar serializações desnecessárias. Um espaço-ator é um conjunto de atores computacionalmente passivo, que representa um contexto de emparelhamento de padrões. *ActorSpace* utiliza um modelo de comunicação baseada em padrões de destino, cada um dos quais está associado com uma lista de atributos de atores. Assim, mensagens podem ser enviadas para todos os membros ou um membro arbitrário do grupo definido pelo padrão, tornando a comunicação mais eficiente.

4.8.2. Modelos Orientado a Objetos

A partir de agora, apresentaremos uma abordagem superficialmente similar, focada em modelos orientado a objetos, que consiste em estender linguagens orientada a objetos sequenciais, fazendo com que mais de uma *thread* esteja ativa ao mesmo tempo.

A primeira maneira, chamada Orientação a objetos Externa, permite *threads* de controle no nível mais alto da linguagem. Os objetos continuam com sua função de reunir

código logicamente relacionados. Dessa forma, a declaração de um objeto funciona como um mecanismo de comunicação, visto que ela pode ser alterada por um método que está executando em uma *thread*, e observada por um método que está executando em outra *thread*.

Alguns exemplos de linguagens baseadas em modelos de Orientação a objetos Externa são: ABCL/1, ABCL/R, EPL, *Emerald, Concurrent Smalltalk* [24]. Nessas linguagens o paralelismo é baseado na especificação de uma *thread* a cada objeto e no uso de mensagens assíncronas para reduzir bloqueios.

ABCL/1 é a versão orientada a objetos do modelo *Actors*, onde objetos independentes podem executar em paralelo. Seus princípios são praticidade e semântica clara de passagem de mensagem. São definidos três tipos de passagem de mensagem: passado, presente e futuro. Apenas o modo presente opera sincronamente, os outros dois modos são assíncronos. Pra cada um dos três mecanismos de passagem de mensagem, ABCL/1 proporciona dois modos distintos: normal e expresso, que corresponde a duas filas de mensagens diferentes que são processadas serialmente dentre de um objeto. Além disso, o paralelismo entre objetos é limitado, porque, em um mesmo objeto, pode chegar no máximo apenas uma mensagem. ABCL/R é uma extensão do ABCL/1, mas que não vamos abordar neste capítulo.

A segunda maneira, chamada de Orientação a objetos Interna, encapsula o paralelismo dentro dos métodos de um objeto, mas o nível mais alto da linguagem tem aparência sequencial. Essa técnica é chamada de paralelismo de dados.

O *Mentat Language Programming* (MLP) é um sistema paralelo orientado a objetos projetado para desenvolver aplicações paralelas de arquitetura independente. Esse sistema integra um modelo de computação dirigido a dados com o paradigma orientado a objetos. O modelo dirigido a dados sustenta um alto grau de paralelismo, enquanto o paradigma orientado a objetos esconde boa parte do ambiente paralelo do usuário. MLP é uma extensão de C++. O compilador e o apoio em tempo de execução da linguagem são planejados para alcançar alto desempenho. Os programas em *Mentat* são representados como grafos dirigidos. Os vértices são elementos de computação que executam algumas funções e as arestas modelam dependência de dados entre esse elementos. O compilador gera código para construir e executar as dependências de dados do grafo. Deste modo, paralelismo inter-objetos em *Mentat* é bastante transparente para o programador. Por exemplo [24] , sejam A, B, C, D, E vetores e considere as instruções:

$$A = vect_op.add(B,C);$$

$$M = vect\_op.add(A, vect\_op.add(D,E));$$

O compilador e o sistema em tempo de execução detectam que as duas somas $B+C$ e $D+E$ não têm dependência de dados e podem ser executadas em paralelo. Então, o resultado é automaticamente encaminhado para o final da soma.

4.8.3. Arrays sistólicos

Um *Array Sistólico* é composto de uma matriz de Unidades de Processamento de Dados (DPU) chamados de células. Essa matriz corrresponde a uma arquitetura em grade de

células (Figura 4.3). DPU são similares a Unidades de Processamento Central (CPU). Cada célula compartilha a informação com suas vizinhas imediatamente depois do processamento. Analogamente às dinâmicas sistólicas do coração, computadores sistólicos realizam operações de maneira rítmica, incremental e repetitiva, e passam dados para células vizinhas ao longo de uma ou mais direções. Em particular, cada elemento computa um resultado incremental e o computador sistólico deriva o resultado final interpretando os resultados parciais de todo o *array*.

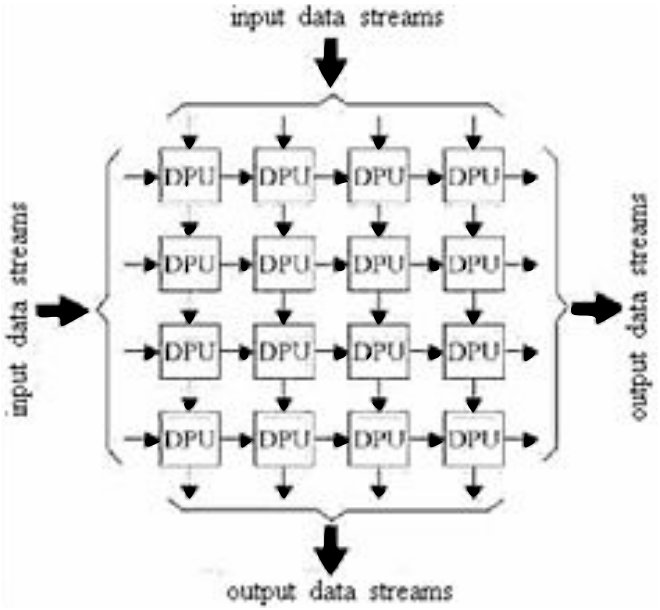


Figura 4.3. Array Sistólico. Fonte: [27].

Um programa paralelo com um *array* sistólico deve especificar como as informações são mapeadas nos elementos sistólicos e o fluxo de informações através dos elementos. *Arrays* de alto nível programáveis permitem o desenvolvimento de algoritmos sistólicos por meio da definição de paralelismo inter e intracélulas, e a comunicação entre as células. O princípio de comunicação rítmica diferencia *arrays* sistólicos de outros computadores paralelos, e a programabilidade de alto nível deste modelo proporciona flexibilidade da arquitetura sistólica. Um exemplo é a linguagem *Alpha*, onde programas são expressados como equações de recorrência. Elas são traduzidas para a forma sistólica, definindo-se um espaço vetorial que pode ser transformado geometricamente, e levando em consideração as dependências de dados.

Um exemplo de aplicação: avaliação polinomial.

A regra de Horner para avaliar um polinômio é:

$$y = (...(((a_n * x + a_{n-1}) * x + a_{n-2}) * x + a_{n-3}) * x + ... + a_1) * x + a_0$$

Um *array* sistólico linear em que os processadores são arranjados em pares: um multiplica sua entrada por x e passa o resultado para a direita, e o próximo soma a_j e passa o resultado para a direita (para o próximo par).

Vantagens: rápido e escalável.

Desvantagens: caro, difícil de construir e altamente especializado para aplicações muito particulares.

4.9. Tudo Explícito

Nesta categoria encontram-se técnicas, modelos e linguagem em que os detalhes da decomposição e comunicação são explícitos e, portanto, a aplicação é específica para uma arquitetura e o estudo é dificultado pela necessidade de grande detalhamento no código. Esta é a classe que corresponde à primeira geração de computação paralela.

4.9.1. Paradigmas de particionamento, mapeamento e comunicação

Message passing, shared memory (memória compartilhada), *rendezvous e data parallel* são paradigmas de particionamento, mapeamento e comunicação que garantem o gerenciamento transparente da decomposição e comunicação em modelos explícitos, sendo peças chaves para o início do entendimento desse tipo de arquitetura.

No paradigma *message passing* são oferecidos mecanismos para troca de informação de processos necessitando da utilização de sistemas de suporte *run-time* para lidar com o uso de aplicações, alocação de processos, trocas entre mensagens e objetos. São utilizadas operações de *send e receive* para identificar processos, endereços e mensagens. Em sistemas de memória distribuída, o *message passing* permite que sejam implementados programas eficientes. Tais programas demandam detalhamento na paralelização, porém este é o modelo mais eficiente e que também fornece suporte a outros modelos, inclusive de mais altos níveis.

Em comunicação por *shared memory*, assim como nas técnicas de sistemas operacionais com memória compartilhada, são utilizadas variáveis compartilhadas e semáforos. Entretanto, troca-se a multiprogramação pelo multiprocessamento verdadeiro. Como o programador precisa realizar a comunicação, sincronização e mapeamento de processos, particionamento, distribuição e entrada e saída de dados há maior dificuldade na exploração do paralelismo e no aprendizado, sendo estes sistemas caracterizados como de baixo nível.

No caso do estudo de aplicações numéricas científicas, destaca-se o paradigma *data parallel* que distribui os vetores por múltiplos processadores, e então os cálculos são atribuídos de acordo com o vetor de elementos mais relevante para o processador. Um pré-processador gera o código de distribuição de dados, que varia de acordo com a plataforma, a partir de diretivas em programas implementados em C ou Fortran. *Data parallel* pode ser implementada com MPI[4] (*Message Passing Inteface*) e em sistemas de memória compartilhada com bibliotecas de *threads*. Um exemplo de linguagem que usa este paradigma é o *High Performance Fortran* (HPF) que integra diretivas *data-parallel* na linguagem do *core* e do compilador.

Outro tipo de comunicação utilizado em sistemas de memória distribuída é o *rendezvous* [7]. No *rendezvous*, um processo faz uma chamada de *entry* (uma requisição) para um outro processo que deve executar uma chamada de *accept* para tal *entry* (devolver

[4] seção 4.10.3

uma resposta). Se o processo que deve executar o *accept* não puder fazê-lo, o processo que efetuou a requisição entra em uma fila e é suspenso até que se possa dar o *accept*. Assim, processos trocam parâmetros e executam regiões críticas, e os mecanismos de exclusão mútua e sincronização de tarefas reduzem o número de primitivas necessárias. As linguagens *Ada* e *Concurrent C* utilizam *rendezvous* e geralmente não são linguagens familiares ao estudante de programação, uma vez que *Ada*, por exemplo, foi desenvolvida para programas em tempo real e é utilizada na aviação e em naves espaciais [19].

4.9.2. Modelos

Na decomposição e comunicação explícitas, podem ser utilizados paradigmas específicos ou gerais, que são de grande importância no ensino de tais modelos. A maioria utiliza paradigmas específicos, pois com a utilização de paradigmas gerais, apesar de se obter desempenho garantido e medidas de custo, são exigidos maiores detalhes que dificultam a programação e, portanto, exigem maiores esforços no aprendizado.

O MPI (*Message Passing Inteface*), por exemplo, é uma ferramenta que faz uso de *message passing* (troca de mensagens) no desenvolvimento de programas paralelos. O MPI pode ser aplicado às linguagens C, C++, que frequentemente são utilizadas em ambientes de ensino, além de Fortran 77 e 90 e é usualmente empregado em computação de alto desempenho. Existem muitas implementações e quase todo *hardware* possui suporte para MPI, e muitos programas que o utilizam podem ser executados em diferentes plataformas sem a necessidade de se reescrever código.

Em *Ada* o paralelismo é baseado em processos chamados *tasks* que se sincronizam por *rendezvous*. A linguagem é imperativa e orientada a objetos. O seu compilador é capaz de otimizar a saída do programa, mas para tanto requer maiores informações sobre o código [19].

Como exceção, algumas das linguagens que utilizam múltiplos paradigmas são a PI [9], a *Orca* [14] e a SR [4] (*Synchronizing Resources*). *Orca* é uma linguagem baseada em objetos, na qual processos pai podem passar objetos para seus filhos estabelecendo uma comunicação. A comunicação é possível inclusive em máquinas diferentes. A linguagem SR fornece suporte a variáveis compartilhadas por processos, mensagens assíncronas, *rendezvous* e *multicast*. Os processos são agrupados em módulos chamados recursos que são dinamicamente criados e que realiza comunicação por meio de semáforos. Diferentes recursos se comunicam por operações definidas explicitamente como procedimentos.

4.9.3. Java

Assim como as demais linguagens interpretadas, Java usa *shared memory* baseada em *threads* que se comunicam e sincronizam com variáveis de condição [24]. As API's (*Application Programming Interface*), JNI's (*Java Native Interface*) e *sockets* facilitam a extensão, pois fornecem as funcionalidades, os padrões e as ferramentas necessárias. As *Java Virtual Machines* (VM) garantem portabilidade, já que o único requisito necessário é uma VM adequada ao sistema operacional. Por isso, Java é um importante exemplo de linguagem em que o usuário especifica a comunicação e a decomposição.

Java Virtual Machines (VM)

As VM's de Java são responsáveis pela interpretação dos *bytecodes* e dividem seus sistemas paralelos em ambientes *single virtual machine (single-VM) e multiple virtual machine (multi-VM)*. Em ambientes *single-VM* uma única máquina virtual executa uma única aplicação em várias *threads* e em ambientes *multi-VM* várias máquinas virtuais executam aplicações diferentes, em processadores diferentes ou não, que se comunicam umas com as outras. Assim o single-VM é mais adequado para arquiteturas de *hardware* que possuem um único espaço de endereçamento (*single system image*), e o *multi-VM* é mais adequado a computadores MPP (*Massive Parallel Processing*) ou *clusters Beowulf* [6].

Em ambientes *single-VM*, pode-se utilizar o modelo de *threads* de Java como paradigmas de programação paralela sem grandes modificações. É necessário apenas garantir número de *threads* suficientes para ocupar os processadores. Em implementações mais recentes de VM's, as classes de *threads* são implementadas de tal forma que o sistema operacional é capaz de enxergá-las e alocá-las nos diferentes processadores facilitando o acontecimento do processamento paralelo.

Apesar destas facilidades, para escrever programas paralelos com as *threads* em Java a forma de raciocínio que deve ser utilizada não é a mesma que se aplica nos casos de programas concorrentes ou sequenciais. Deve-se se manter o número de *threads* igual ao de processadores, administrando a criação e atividade específica a cada uma delas, de forma a se aproveitar a capacidade do processador, pois criar *threads* é uma atividade custosa. A falta de existência de sincronização de barreira para a classe *thread* é mais uma dificuldade a ser superada na produção de programas paralelos.

Quando há grande quantidade de processadores, fatores como otimização de comunicação e localidade interferem na eficiência, então *single-VM* deixam de ser atraentes e *multi-VM* se tornam preferíveis. Contudo, *multi-VM* ainda necessitam do desenvolvimento de API's que gerenciem o mapeamento das *threads* na memória. No momento, API's tem sido adaptadas para realizar a comunicação entre VM's, bem como bibliotecas de outras linguagens para comunicação entre processadores e extensões incluindo recursos novos.

A Java RMI (*Java Remote Method Invocation*) [2] é uma API que tem sido adaptada e que permite métodos remotos de chamada em objetos em diferentes MV's em uma rede, além de aplicações distribuídas de objetos Java. Também constrói camadas que escondem os detalhes de comunicação ao nível de chamada de métodos do desenvolvedor. Essas características da Java RMI garantem a solução de problemas cliente-servidor, em que um servidor cria objetos que invocam métodos nesses objetos remotamente [6]. Essa solução não é útil em aplicações de alto desempenho, nas quais, geralmente, há comunicação entre processos que fazem uma cópia local dos dados para a realização dos cálculos necessários. Para esses problemas parece não ser viável a adaptação.

Aliado a essa dificuldade há problemas de desempenho, serialização de objetos Java (transformação de objetos e referências em estruturas que facilitem a comunicação) e do RMI. A adaptação tem resultado em processos custosos quando comparado com outros mecanismos o que parece limitar sua aplicabilidade.

Com relação a adaptações de bibliotecas não nativas, o MPI, apesar de não ser orientado a objetos, pode ser implementado em Java com, por exemplo, *socket programming*, RMI, JNI, Java-toc interface ou PVM. Como exemplo de implementação temos o NPAC's mpiJava que utiliza o modelo de ligação MPI C++ como base e interfaces para diferentes implementações nativas de MPI.

Desta forma, no paralelismo em Java há a questão de comunicação entre *threads* e entre VM's, o que pode possibilitar a construção de *softwares* mais complexos. Para a obtenção de ainda maior eficiência do modelo seriam necessárias JNI's eficientes para a comunicação entre processadores.

Portanto, em aplicações paralelas, o modelo nativo atual de *threads* provavelmente limitará a capacidade de programação. No entanto, modificar o modelo ou utilizá-lo em conjunto com outro modelo de comunicação pode ser complicado.

4.9.4. Paralelização de laços

Uma das aplicações mais óbvias do paralelismo em códigos sequenciais é a paralelização de laços. Como a forma mais comum de se explorar o paralelismo em Java com *threads* é ao nível de método, para a execução de um laço em paralelo é necessário criar um método com o laço, calcular ligações de laço para cada *thread* e passar uma instância da classe do método para o administrador de tarefas para execução em cada *thread*, o que também ocorre com códigos em C ou Fortran com a biblioteca de *threads* POSIX. Esse processo é muito trabalhoso, mas algumas partes podem ser automatizadas usando diretivas de compilador. Antigamente havia complicações pelo fato de que estas diretivas eram específicas de cada fabricante, porém esforços têm sido feitos em relação à criação de padrões.

4.9.5. Paralelização automática

Uma abordagem diferente para o paralelismo vem sendo abordada com a biblioteca de *threads* em arquiteturas de *shared memory*, na qual se tenta atingir o processamento de códigos sequenciais para paralelização automática pelo compilador. A paralelização do código envolve análise de dependência de dados e condições de concorrência que dificultam o procedimento mesmo considerando apenas laços. São necessárias estratégias de transformação de laços, de laços de corpos extensos e com chamadas de procedimento. Além disso, transformações aplicadas a laços são essencialmente locais para cada laço, visto que a estratégia para otimizações globais para localidade de dados são extremamente difíceis de implementar [6].

4.9.6. Modelos com Tudo Explícito: Síntese

Java é uma linguagem portável e que só necessita de uma *Java Virtual Machine* apropriada. Seus modelos de *threads* que se comunicam e sincronizam são adequados ao paralelismo, mas ainda são necessários desenvolvimentos ou adaptações de recursos como a comunicação, sincronização e distribuição de processos que são complexos e podem levar ainda algum tempo.

Ada, PI, *Orca* , SR e Java são linguagens que permitem a criação dinâmica de processos e comunicação. Diferentemente, *Occam* fixa a estrutura dos processos, que se

comunicam por canais assíncronos. Para tanto, são utilizadas as primitivas *assignment*, *input* e *output* [24] que podem ser combinadas com construtores paralelos, inclusive não determinísticos. Sua aplicação fica restrita a aplicações críticas ou pequenas por causa de seu baixo nível.

4.10. Mais Modelos

Os modelos e linguagens descritos e estudados nesta seção não foram incluídos na classificação acima, apesar de serem igualmente importantes aos métodos apresentados até aqui. Estes modelos possuem características que merecem destaque ou porque correspondem a padrões superiores para a paralelização, mesmo sendo apenas teóricas e não tão modernas – como é o caso de PRAM –, ou porque, no sentido inverso, representam os avanços mais recentes e podem indicar tendências para o futuro.

4.10.1. PRAM

Parallel Random Access Machine, ou PRAM como é mais comumente conhecido, consiste em um dos modelos de computação paralela mais amplamente estudados, servindo de base para diversos outros modelos, tais como APRAM (*Asynchronous Parallel Random Access Machine*), LPRAM (*Local-Memory Parallel Random Access Machine*) [16].

Devido sua ampla importância conceitual no estudo e desenvolvimento de algoritmos eficientes, PRAM se apresenta como um dos modelos mais importantes quando se tem como objetivo o ensino ou aprendizado de computação paralela pois a teoria na qual ele se baseia dá enfoque à resolução de problemas paralelos e seus algoritmos. Entretanto, alguns pontos presentes nesta teoria tornam PRAM um modelo de difícil implementação prática (como veremos mais adiante).

Este modelo consiste em um conjunto de processadores trabalhando de forma sincronizada (Figura 4.4), ou seja, podem executar instruções independentes mas são sincronizados antes de cada instrução, processadores esses que possuem cada um uma memória local própria e uma memória global comum a todos. Esta memória global é compartilhada entre o conjunto de processadores, sendo que existem diferentes protocolos para organizar o acesso.

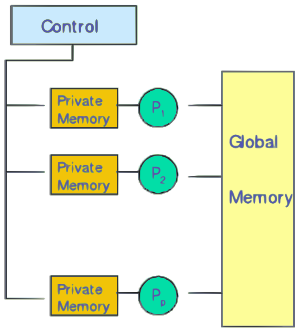


Figura 4.4. Estrutura do modelo PRAM. Fonte: [15].

São alguns deles:

- EREW (*Exclusive Read, Exclusive Write*): Conflitos de leitura ou escrita não são permitidos;
- CREW (*Concurrent Read, Exclusive Write*): Permite a leitura ao mesmo tempo. Mas conflitos de escrita não são permitidos. Este é o modelo PRAM padrão;
- CRCW (*Concurrent Read, Concurrent Write*): Permite a leitura e a escrita ao mesmo tempo por processos diferentes;

Quando pensamos em escrita concorrente (*concurrent write*) vemos que surge o problema de conflitos de escrita, ou seja, mais de um processador escrevendo no mesmo local da memória compartilhada ao mesmo tempo. Existem diferentes políticas de tratamento para esses casos. Algumas delas são:

- Arbitrário: escolhe arbitrariamente aquele que irá escrever naquele local da memória;
- Prioridade: assume que existe uma hierarquia de prioridades entre os processadores e aquele com maior prioridade ganha;
- Combinacão: combina os valores de alguma forma;
- Detecção: algum valor especial é escrito quando há uma tentativa concorrente de escrita.

O modelo PRAM assume como verdade certos aspectos a cerca da arquitetura do computador com o propósito de prover uma maior abstração e facilitar o desenvolvimento de algoritmos neste contexto. Algumas das conjecturas feitas por esse modelo são:

- há um número ilimitado de processadores;
- há uma quantidade ilimitada de memória;
- o custo de acesso a memória é sempre uniforme;
- o custo de sincronização dos processadores ao final de cada ciclo é também dito como uniforme.

Porém, apesar de facilitarem o desenvolvimento de algoritmos, a medida que ajuda a manter o foco na semântica do problema e diminuem as preocupações com detalhes de arquitetura, estas conjecturas dificultam análises mais profundas do algoritmo desenvolvido, tais como o tempo total de execução e utilização de recursos da máquina, o que por sua vez torna mais complexo o processo de corretude de algoritmos desenvolvidos sob este modelo.

Tendo isto em vista, foram desenvolvidos modelos baseados no modelo PRAM (como alguns já citados acima) que tem como preocupação principal manter a abstração provida pelo modelo PRAM porém introduzindo mais detalhes sobre a arquitetura na

tentativa de se aproximar da realidade das máquinas atuais. Foram desenvolvidas algumas linguagens neste modelo tais como Fork95 e a XMTC.

Fork95 é apresentado em seguida, pois além de um exemplo prático do que pode ser feito baseado no modelo em questão pode-se encarar como um incentivo ao estudo do modelo.

Fork95 é uma linguagem desenvolvida para a arquitetura SB-PRAM1, com base no modelo PRAM. Desenvolvida pelo Prof. Dr. Christoph Kessler[5] da Linköping University e pelo Prof. Dr. Helmut Seidl[6] da Technische Universität München ano de 1994, é uma extensão da linguagem ANSI C, adicionando a esta rotinas para o suporte ao desenvolvimento de programas paralelos.

Tem como filosofia distribuir as *threads* entre todos os processadores disponíveis desde o começo da execução, calculando o "tamanho das tarefas" e distribuindo com base nisso. A linguagem introduz o conceito de grupo, ou seja, os processadores disponíveis são divididos em grupos e, dentro de seu respectivo grupo, todos os processadores executam suas tarefas de modo síncrono, sendo que é possível existirem subgrupos dentro de grupos, criando uma hierarquia.

4.10.2. Linguagem Erlang

Erlang[7] consiste de uma linguagem de programação desenvolvida nos laboratórios da Ericsson em meados dos anos 80. Existem duas teorias para a origem do nome *Erlang*: a primeira defende que foi dado em homenagem ao matemático dinamarquês Agner Krarup Erlang, autor de muitas contribuições científicas que se aplicam principalmente ao campo da telefonia[8]. Já a segunda, justifica pela combinação de *Ericsson Language*.

Erlang nasceu de uma busca por uma nova ferramenta que combinasse elementos de linguagens já conhecidas com outros elementos que suprissem as necessidades características de sistemas de telecomunicação, tais como lidar com inúmeras tarefas concorrentes, um sistema confiável de tratamento de erros e possibilidade de manutenção em partes do código sem a necessidade de afetar todo o sistema.

Visando conseguir tal objetivo, uma equipe de profissionais da Ericsson liderada pelo cientista da computação Joe Armstrong [3] realizou experimentos com base na linguagem Prolog, dando origem a linguagem em questão. Desde então, *Erlang* vem ganhando espaço, sendo utilizada em muitos projetos da Ericsson e por inúmeras outras empresas (como o *Facebook chat backend*) e também por universidades após o lançamento de versões *open source*.

Devido a preocupação com a portabilidade de sistemas escritos em *Erlang*, foi implementada uma máquina virtual, que funciona como as já conhecidas máquinas virtuais para uma linguagem interpretada: um código escrito em *Erlang* é compilado em *byte codes* que por sua vez são interpretados pela máquina virtual. Desta forma, desde que haja uma máquina virtual *Erlang* instalada na plataforma, não haverá problemas de

[5]http://www.ida.liu.se/ chrke/
[6]http://www2.in.tum.de/ seidl/
[7]http://erlang.org/
[8]http://www.erlang.com.br/brhistor.asp

compatibilidade, o que torna a linguagem altamente portável e retira a necessidade de implementar versões para arquiteturas específicas.

Dada a natureza das aplicações para as quais a linguagem foi inicialmente projetada (sistemas de telecomunicação), *Erlang* apresenta um poderoso suporte a concorrência. Cada máquina virtual *Erlang*, também chamada de nó, pode criar processos que são executados em paralelo e se comunicam através de um sistema assíncrono de mensagens. Cada mensagem recebida é colocada numa fila de mensagens, sendo posteriormente consumida pelo processo receptor, não havendo portanto uso de memória compartilhada. Assim, se ocorre um erro em tempo de execução, apenas o processo onde ocorreu o erro é terminado, garantindo a segurança de outros processos.

Outro aspecto interessante da linguagem reside na possibilidade de substituição de partes do código em tempo de execução do sistema, o que também é conhecido como *hot swapping*. Isto é possível, entre outros fatores pois uma aplicação em *Erlang* não é carregada inteiramente desde o princípio de sua execução mas sim aos poucos, conforme determinadas partes do código vão sendo requisitadas, o que é chamado de carregamento incremental. Assim, torna-se possível a manutenção e substituição de partes do código em sistemas ainda em execução, sendo que esta característica é de extrema importância quando se trata de aplicações em telefonia, *web services* e outros serviços vitais que demandam alta confiabilidade e manutenção rápida e eficiente ou ainda sistemas que possuem extensas entradas de dados e cálculos exaustivos tais como nas áreas de bioinformática e meteorologia.

4.10.3. Message-Passing Interface

O *Message Passing Interface* (MPI) é uma especificação padrão que oferece infraestrutura para a comunicação de dados entre muitos computadores. Por si só, MPI não é uma biblioteca, mas é a especificação de como deve ser uma biblioteca para este tipo de comunicação. Este padrão foi criado entre os anos de 1992 e 1994, e é baseado em convenções da *MPI Forum*[9], que possui mais de 40 organizações participantes, entre fornecedores, pesquisadores, desenvolvedores e usuários [13].

O objetivo do MPI é estabelecer um padrão para programas com passagem de mensagens que seja portável, eficiente, flexível. MPI não é um padrão IEEE ou ISO, mas corresponde a um padrão industrial para o desenvolvimento de programas em HPC (*High-Performance Computing*).

A maioria das implementações de MPI consistem de um conjunto de rotinas disponíveis principalmente para C, C++ ou Fortran. As implementações mais conhecidas são MPICH[10], LAM/MPI[11] e, mais recentemente OpenMPI[12], todas gratuitas e com código aberto[10].

Uma aplicação implementada com MPI é constituída por processos que se comunicam, e que acionam funções para o envio e recebimento de mensagens entre eles.

[9] www.mpi-forum.org
[10] www.mcs.anl.gov/research/ projects/mpich2
[11] www.lam-mpi.org
[12] www.open-mpi.org

Geralmente, um número fixo de processos é criado inicialmente. Cada um desses processos podem executar diferentes programas. Por isso, o padrão MPI é algumas vezes referido como MPMD (Multiple Program Multiple Data). MPI suporta tanto a comunicação ponto-a-ponto quanto a coletiva, ou seja, os processadores podem efetuar operações para enviar mensagens de um determinado processo a outro, ou um grupo de processos pode efetuar operações de comunicações globais. Além disso, MPI é capaz de suportar comunicação assíncrona e programação modular, através de mecanismos de comunicadores que permitem ao usuário MPI definir módulos que encapsulem estruturas de comunicação interna.

Entre todas as vantagens do MPI, estão:

- Portabilidade: Pode-se colocar uma aplicação em uma plataforma diferente sem alterar o código, desde que a nova plataforma suporte o padrão MPI;
- Disponibilidade: Há uma enorme variedade de implementações disponíveis;
- Funcionalidade: Mais de 300 rotinas estão definidas em MPI;
- Padronização: É a única especificação de passagem de mensagens considerada um padrão. É suportada virtualmente por todas as plataformas HPC. É capaz de substituir praticamente todas as bibliotecas anteriores.

Existem duas versões do MPI que são populares atualmente: MPI-1, que dá ênfase a passagem de mensagem e tem um ambiente estático de execução; e o MPI-2 (concluída em 1996), que inclui novas características tais como I/O paralelizável (*Input/Output*), gerenciamento dinâmico de processos e operações de leitura e escrita em memória remota [10]. Além disso, MPI-2 especifica mais de 500 funções e oferece suporte para implementações em ANSI C, ANSI C++ e ANSI Fortran (Fortran90).

É importante ressaltar que MPI-2 foi criado de modo que o MPI-1 seja um subconjunto do MPI-2 – apesar de algumas funções estarem *deprecated*. Assim programas em MPI-1 continuam funcionando em implementações de MPI-2.

O principal destaque na especificação de MPI-2 é a gerenciamento dinâmico dos processos, pois é permitido a criação (possivelmente em máquinas distintas) e a finalização cooperativa (isto é, de modo que a integridade do sistema seja garantida) de processos mesmo depois de a aplicação ter começado sua execução. Além disso, permite estabelecer comunicação entre processos que foram disparados separadamente [17].

Outra nova funcionalidade é a chamada *One-sided Communications*, que são rotinas para a comunicação unidirecional. Inclui operações com *put*, *get* e *accumulate*, que possibilitam ler e escrever em memória remota e operações de redução através de algumas tarefas (*tasks*). Exitem também operações de coletivas e de sincronização, fazendo com que, em ambientes de memória compartilhada, este modelo apresente-se mais conveniente do que troca de mensagens [10].

Uma coleção de funções também estão disponibilizadas no MPI-2 – chamadas de I/O Paralelo – que foram projetadas para permitir abstração das dificuldades na manipulação de I/O (entrada e saída) em sistemas distribuídos. Também possibilitam o fácil acesso

a arquivos por meio de uma estrutura padronizada. Entretanto, algumas pesquisas que vêm sendo feitas sobre essa ferramenta mostra que ela não tem bom desempenho. Por exemplo, implementações paralelizadas para multiplicação de matrizes esparsas usando a biblioteca de I/O do MPI apresentam resultados de eficiência muito ruins.

Além dessas inovações, MPI-2 apresenta extensões de comunicações coletivas, operações coletivas não-bloqueantes, aplicação de operações coletivas para intercomunicadores.

4.10.4. CORBA

Ainda no contexto de ambientes distribuídos, apresentamos o modelo CORBA[13] cujo diferencial é a orientação a objetos. *Common Object Requesting Broker Architecture* é a arquitetura padrão proposta pela OMG (*Object Management Group*) para estabelecer e simplificar a troca de dados entre sistemas distribuídos heterogêneos.

A OMG é uma organização formada por empresas dos diferentes ramos da informática: desenvolvimento, produção, venda. Inicialmente somavam-se treze empresas, hoje já são setecentos e cinquenta, entre elas estão: Cannon, IBM, Sun, Apple, e outras. Essas empresas trabalham sem fins lucrativos, para promover a criação e elaboração de modelos e padrões que proporcionem a interoperabilidade entre aplicações que usam tecnologia orientada a objeto. Dessa maneira nasceu a tecnologia CORBA [20].

Diante da diversidade de *hardware* e *software* que encontramos atualmente e da dificuldade de compatibilidade, a CORBA atua de modo que seus objetos (os componentes de *software* que são disponibilizados pelo padrão) possam se comunicar de forma transparente para o desenvolvedor, mesmo que para isso seja necessário interoperar com outro *software*, em outro sistema operacional e em outra ferramenta de desenvolvimento – interoperabilidade local ou remota entre aplicações. CORBA é um dos modelos mais populares de objetos distribuídos, juntamente com seu maior concorrente, o DCOM (*Distributed Component Object Model*), tecnologia proprietária da Microsoft.

O surgimento deste padrão é datado de 1991, mas a versão 2.0 foi disponibilizada em 1994.

CORBA utiliza a IDL (*Interface Definition Language*): uma linguagem baseada em C++ e usada para definir a interface de um objeto, que é composta pelas operações que esse objeto oferece, assim como os seus tipos de dados. Diferentemente de linguagens de programação, a IDL não tem comandos condicionais ou de repetição (é uma linguagem puramente declarativa). Logo, os objetos não são implementados em IDL, mas apenas definidos. Isso garante a independência da interface dos objetos de qualquer linguagem de programação. A IDL dá suporte a tipos simples (números inteiros, caracteres, *strings*), a tipos complexos (enumerações, exceções) e até a herança múltipla. Existe também uma herança *default*: todos os objetos definidos em IDL herdam implicitamente da interface *Object*, própria do CORBA. Ela define operações comuns a todos os objetos CORBA [20].

Além de uma linguagem de definição de interfaces, CORBA oferece o mapeamento desse código declarativo para a linguagem de programação utilizada para o desen-

[13] http://www.corba.org/

volvimento de clientes e servidores.

Outra funcionalidade importante de CORBA é o ORB (*Object Request Broker*), um conjunto de módulos que gerenciam a comunicação entre objetos. Alguns autores o definem com um barramento de objetos. Com ele, objetos podem fazer requisições a outros objetos que podem estar localizados localmente ou remotamente, de forma transparente. Essa transparência assegura que o cliente (objeto requisitante) não tenha conhecimento de quais mecanismos utilizados na comunicação com o objeto requisitado.

4.10.5. OpenMP

Open Multi-Processing (OpenMP[14]) é uma API desenvolvida para sistemas que utilizam memória compartilhada, com suporte às linguagens C, C++ e Fortran e também a diferentes arquiteturas, como Unix e Windows. Além de portabilidade, tem como objetivos apresentar um modelo escalável e fornecer ao programador uma interface simples e flexível [10].

Por meio de *threads* que são criadas a partir de uma única *thread* principal, o programador pode manipular como será executado seu código, especificando quais os trechos que devem ser paralelizados e quais variáveis deverão ser compartilhadas. Tais informações são fornecidas através de diretivas para o compilador, semelhantes a comentários. Segue abaixo um pequeno exemplo de código em que é feita a inicialização de um vetor por meio de diversas *threads*:

```
1  int main(int argc, char **argv) {
2       const int N = 100000;
3       int i, a[N];

4       #pragma omp parallel for
5       for (i = 0; i < N; i++)
6           a[i] = 2 * i;
7      return 0;
8  }
```

As principais vantagens de OpenMP em relação aos outros modelos residem na despreocupação com sincronização de mensagens (em relação à interface MPI), a flexibilidade mesmo em relação à aplicações sequenciais (um compilador comum interpretará as diretivas OpenMP apenas como comentários) e a possibilidade de paralelizar diversos trechos de código sem modificá-los estruturalmente (diminuindo a incidência de erros de programação).

Porém a interface ainda não apresenta alguns recursos desejáveis, como um verificador de erros confiável, garantia de máxima eficiência no uso da memória compartilhada e sincronização automática de entrada e saída de dados[15].

Encontra-se atualmente na versão 3.0 (2008) e é administrada pela comunidade OpenMP ARB (*OpenMP Architecture Review Board*), composta por diversas empresas

[14]http://openmp.org/

[15]https://computing.llnl.gov/tutorials/openMP

interessadas de alguma forma no contínuo desenvolvimento desta interface, entre elas: AMD, IBM, Sun Microsystems e Intel[16].

4.10.6. Modelos de balanceamento de carga e TBB

Na seção 4.4 foi apresentada a grande dificuldade de se paralelizar um código sequencial sem que houvesse informações explícitas por parte do programador, indicando os trechos desejados para a paralelização e/ou como os processos deveriam se comunicar entre si, por exemplo. Porém, atualmente, algumas linguagens estão sendo desenvolvidas com o objetivo de tornar cada vez mais eficiente esta transformação, visando não somente uma paralelização automática de código, mas também economizar ao máximo recursos e minimizar o tempo total de execução. Entre elas, podemos citar a linguagem Cilk[17] e as bibliotecas Kaapi[18] e TBB[19], sendo esta última apresentada ainda nesta seção. O ponto em comum entre essas três abordagens é a utilização de modelos de balanceamento de carga para escalonamento dinâmico de tarefas.

No escalonamento dinâmico, a partir de estimativas dos custos das tarefas das aplicações, elas são alocadas a recursos. Mas, nesse caso, como o ambiente é completamente dinâmico, costuma-se permitir a alocação de tarefas no momento de sua criação. Isto é feito através de mecanismos de balanceamento de carga: os recursos menos carregados recebem as tarefas [10].

Assim, em tempo de execução, o escalonamento é revisto com base nas condições atuais dos processos, alterando as alocações se julgar necessário, o que não é possível no escalonamento estático. O uso deste tipo de mecanismo objetiva o aproveitamento máximo dos recursos disponíveis, deixando-os ocupados o maior tempo possível[20].

Neste contexto, a Intel desenvolveu a biblioteca *Threading Building Blocks* para C++ possibilitando implementações de programas que utilizem os diversos núcleos dos atuais processadores *multi-core*. Seu modelo de balanceamento é chamado de *task stealing* e funciona basicamente da seguinte maneira: se um dos núcleos já completou o seu trabalho enquanto outros possuem uma razoável quantidade de tarefas em suas respectivas filas, a TBB rearranja estas tarefas, retirando algumas de uns dos núcleos ocupados e repassando para o que está livre. É importante ressaltar que todo este processo ocorre sem a requisição ou manipulação do programador, que pode concentrar-se somente em sua implementação.

Dentre as componentes disponíveis na TBB, destacam-se os *Containers* que se aplicam à exclusão mútua *(concurrent_queue, concurrent_vector, concurrent_hash_map)*, as operações atômicas *(fetch_and_add, fetch_and_store, compare_and_swap)* e o *Task Scheduler*, que possibilita o acesso direto a criação e ativação de tarefas. A TBB encontra-se na versão 2.1, lançada em julho de 2008.

[16] http://openmp.org/wp/about-openmp
[17] http://supertech.csail.mit.edu/cilk/
[18] http://kaapi.gforge.inria.fr/
[19] http://www.threadingbuildingblocks.org/
[20] https://computing.llnl.gov/tutorials/parallel_comp/#DesignLoadBalance

4.10.7. Processamento em GPU e CUDA

As unidades de processamento gráfico, mais conhecidas como GPUs (do inglês *Graphics Processing Units*), são centrais responsáveis por executar instruções descritas por um dado programa de computador, assim como as CPUs (*Central Processing Units*) tradicionais. Porém, dado seu alto poder de processamento e uma série de otimizações, as GPUs são muito utilizadas na manipulação de imagens e gráficos em diversas áreas, tais como simuladores cada dia mais realistas, cinema e outras áreas de entretenimento, em sua maioria na indústria de jogos.

São encontradas em dispositivos específicos para processamento de vídeo tais como placas de vídeo ou ainda nas chamadas *mother-board* (placas-mãe) de certos computadores, porém neste último com um poder reduzido.

Quanto a sua organização interna, existem algumas variações dependendo de sua finalidade principal ou fabricante. Podem possuir memória própria, o que facilita acesso aos dados e otimiza as operações, ou podem compartilhar memória com a CPU da máquina, o que pode causar perda de desempenho.

Sua arquitetura altamente paralela e seu poder de processamento mencionado anteriormente abrem um vasto leque de opções para o uso das GPUs, fazendo desta forma com que se tornem alvo de interesse crescente por parte da academia e da indústria em cálculos matemáticos complexos. Elas trabalham em conjunto com a CPU, se encarregando de cálculos referentes a operações com matrizes e vértices, tranformações lineares e outras operações matemáticas constantemente presentes em aplicações gráficas, deixando a CPU livre para outras tarefas, ganhando em desempenho e qualidade de imagem.

Em consequência disso o uso de GPUs vem se tornando uma tendência e assim foram desenvolvidas algumas linguagens e interfaces de programação para GPUs, facilitando o desenvolvimento de software utilizando esta arquitetura. Temos alguns exemplos importantes como a linguagem *BrookGPU*, desenvolvida por equipes de pesquisa da Universidade de Stanford[21] e CUDA, desenvolvida pela NVIDIA, empresa americana muito conhecida principalmente por produzir *hardware* de aceleração gráfica de alta qualidade.

Compute Unified Device Architecture, ou simplesmente CUDA, é uma arquitetura de computação paralela que oferece uma interface de programação com rotinas e funções que permitem ao programador fazer uso das GPUs presentes nos *hardwares* de aceleração gráfica da NVIDIA. Funciona como uma extensão da linguagem ANSI C, ou seja, programadores utilizam a linguagem C como linguagem principal e conforme necessário fazem uso de rotinas implementadas pela arquitetura CUDA dentro de seu programa, permitindo programação de alto nível ao mesmo tempo em que é possível lidar com rotinas de baixo nível, tais como alocação e liberação de memória.

Como explicado anteriormente, as GPUs funcionam basicamente como um co-processador, trabalhando em conjunto com a CPU principal. Assim, ao programar em CUDA é possível escolher onde uma função será executada através de qualificadores disponibilizados pela linguagem. São eles:

- **_device_ :** funções qualificadas nesta categoria só podem ser chamadas e execu-

[21] http://graphics.stanford.edu/projects/brookgpu/talks.html

tadas pelo dispositivo que contém a GPU, nunca pela CPU central;

- **_host_ :** funções qualificadas com o marcador _host_ só podem ser chamadas e executadas pela CPU da máquina (chamada de *host*), nunca pela GPU. É o valor padrão se nada for declarado;
- **_global_ :** funções desta categoria são chamadas pelo *host* mas são executadas na GPU.

Como CUDA não possui suporte a recursão, funções declaradas como _device_ ou _global_ não podem ser executadas recursivamente, a menos que esteja sendo executado o modo emulado, permitindo executar programas em CUDA mesmo sem a presença de um *hardware* com GPUs. Também possui rotinas para criação de *threads*, sendo que as *threads* são divididas em blocos e identificadas por um número (*threadId*). As *threads* de um mesmo bloco podem ser sincronizadas por uma chamada da rotina _syncthreads_.

4.11. Conclusão

Sugerimos neste capítulo o ensino de métodos paralelos de programação apresentando os diversos modelos e linguagens disponíveis. Foram discutidas características importantes para o aprendizado de desenvolvimento de programas corretos, que exploram bem o paralelismo e com desempenho previsível, como a facilidade de programação, metodologias de corretude, independência de arquiteturas, simplicidade, abstração, garantia de desempenho e de custo.

A partir dessas características, os modelos foram divididos em seis classes, desde modelos mais abstratos, que geralmente satisfazem os critérios do desenvolvimento do *software*, mas não permitem a previsão do desempenho, até modelos mais concretos, para os quais é possível prever o desempenho, mas tornam a construção do *software* complicada.

Ao final, apresentamos modelos e linguagens importantes tanto para a teoria computacional (como PRAM) quanto para a parte prática (como TBB e CUDA). Esses modelos e linguagens não estão inseridos dentro da classificação proposta, mas são de grande influência e destaque nos últimos anos.

Pela descrição e exploração dos modelos é possível observar tendências, como a programação em níveis mais altos e menor detalhamento da paralelização do código provenientes da necessidade de abstração. Outra tendência é a utilização de modelos de abstração intermediária, que procuram esconder alguns detalhes, mas ainda explorando o paralelismo ao máximo gerando programas robustos e com desempenho previsível. Além disso, também é possível observar a crescente necessidade de características como escalabilidade e interoperabilidade dos programas.

A primeira tendência pode influenciar positivamente no ensino, no sentido de facilitar o aprendizado já que o aluno não precisa lidar com os detalhes, ao passo que a exigência do detalhamento faz com que o aluno conheça os mecanismos por trás do paralelismo e entenda melhor o que acontece com seu código.

Alguns modelos, no entanto, fogem desta tendência e conseguem fornecer um

desempenho previsível e satisfatório. Esses modelos parecem mostrar o caminho para o desenvolvimento de linguagens com as características necessárias para construções de bons programas paralelos.

Assim, os avanços no desenvolvimento de modelos paralelos tem convergido no sentido da simplificação e facilitação. Esse tipo de linguagem permite a expansão e maior popularização do paralelismo, dado que abstraem da complexidade da arquitetura.

Referências

[1] AHMED, Shakil; CARRIERO, Nicholas; GELERNTER, David. **A Program Building Tool for Parallel Applications**. Princeton University, 1994 - pages. 161-178.

[2] AL-JAROODI, Jamela; MOHAMED, Nader; JIANG, Hong; SWANSON, David. **An overview of parallel and distributed Jav for heterogeneous systems: approaches and open issues**. Scientific International Journal for Parallel and Distributed Computing, 2002 - Volume 5, n.4.

[3] ARMSTRONG, Joe. **Programming Erlang - Software for a Concurrent World**. Publisher: Pragmatic Bookshelf, 2007.

[4] BAL, Henri E. **A comparative study of five parallel programming languages**. Proc. Of Spring '91 Conf. On Open Distributed Systems, 1991 - pages 209-228.

[5] BORODIN, A.; HOPCROFT, J.E. **Routing, merging and sorting on parallel models of computation**, STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing, 1982 - pages 338-344.

[6] BULL, Mark. and TELFORD, Scott. **Programming Models for Parallel Java Applications**. UKHEC Technical report, 2000 - http://www.ukhec.ac.uk/publications/reports/paralleljava.pdf.

[7] CHANDLER, Angie; FINNEY, Joe. **Rendezvous : An alternative approach to conflict resolution for real time multi-user applications**. 13th Journal Euromicro Conference on Parallel, Distributed and Network-based Processing, 2005 - pages 160-167.

[8] CHIEN, Andrew; KARAMCHETI, Vijay; PLEVYAK, John; ZHANG, Xingbin. **Concurrent Aggregates (CA) Language Report. Version 2.0**. TR, Dep. Computer Science, University of Illinois at Urbana-Champaign, 1993.

[9] DALLY, William; WILLS, Scott. **Universal mechanisms for concurrency**. In: PARLE '89: Proccedings of the Parallel Architectures and Languages Europe, Volume I, 1989. Publisher: Springer-Verlag - pages 19-33.

[10] GOLDMAN, Alfredo; KON, Fabio. **Grades Computacionais: Conceitos Fundamentais e Casos Concretos**. In: Tomasz Kowaltowski e Karin Breitman (Org.) Atualizações em Informática 2008. Editora PUC-Rio. SBC. pp 55-104.

[11] HAMBRUSCH, Susanne E. **Models for parallel computation**. In: Proceedings of Workshop on Challenges for Parallel Processing. International Conference on Parallel Processing, 1996 - pages 92-95.

[12] HEWITT, Carl E. **A Universal, Modular Actor Formalism for Artificial Intelligence**. In: Proceedings of International Joint Conference on Articial Intelligence, 1973.

[13] **High Performance Computing**. Lawrence Livermore National Laboratory, CA (http://computing.llnl.gov/)

[14] KAASHOEK, M. Frans; MICHIELS, Raymond; BAL, Henri; TANENBAUM, Andrew S. **Transparent fault-tolerance in parallel orca programs**. In: Proceedings of the Symposium on Experiences with Distributed and Multiprocessor Systems III, 1992 - pages 297–312.

[15] KRISHNAMURTHY, Arvind. **PRAM Algorithms**. Parallel Computing. Yale University, 2004.

[16] LI, Zhiyong; MILLS, Peter H; REIF, John H. **Models and Resource Metrics for Parallel and Distributed Computation**. In: Proceedings of the 28th Annual Hawaii International Conference on System Sciences, 1989 - pages 133-143.

[17] MPI - Message Passing Interface Forum. **MPI-2: Extensions to the Message-Passing Interface**. November 2003 (http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html)

[18] MOORE, Gordon E. **Cramming more components onto integrated circuits**. In: Readings in computer architecture, 2000. Publisher: Morgan Kaufmann Publishers Inc. pages 56-59.

[19] MUNDLE, David A.; FISHER, David A. **Parallel Processing in Ada**. Domesticating Parallelism, Incremental Systems Corp, 1986 - pages 20-25.

[20] NARDI, Alexandre. **Componentes CORBA**. Dissertação de Mestrado. Instituto de Matemática e Estatística - USP. São Paulo, 2003.

[21] PEYTON-JONES, S. L.; LESTER, D. **Implementing Functional Programming Languages**. International Series in Computer Science, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[22] SACCA, Juliana; KOYAMA, Julio César Hiroshi; TAMAE, Yoshio Rodrigo, MUZZI, Fernando Augusto Garcia. **Processador Cell Broadband Engine**. Simpósio de Ciências Aplicada da FAEF, v. 4, 2007 - pages 431-435.

[23] SILVA, Luís Moura; Buyya, Rajkumar. **Parallel Programming Models and Paradigms**. ACM Computing Surveys, Vol. 30, No. 2, 1998.

[24] SKILLICORN, David B.; TALIA, Domenico. **Models and Languages for Parallel Computation**. In: Journal ACM Computing Surveys, 1998 - pages 123-169.

[25] THEODOROPOULOS, G. K; TSAKOGIANNIS, G. K.; WOODS,J. V. **Occam: an asynchronous hardware description language?**. In: Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology, 1997 - pages 249–256.

[26] WALKER, David; RANA, Omer. **The use of Java in high performance computing: A data mining example**. In: Lecture notes in computer science, Volume 1593, 1999 - pages 861-872.

[27] WIKIPEDIA, the free encyclopedia. **Systolic Array**. (http://en.wikipedia.org/wiki/File:Systolic_array.jpg)

Capítulo

5

Proposta de um Elenco de Disciplinas de Pós-Graduação para Formação de Arquitetura de Computadores

Sergio T. Kofuji, Jussara M. Kofuji, Márcio Lobo Netto, Wang J Chau, Filippo Valiante Filho, Leonardo A. G. Garcia, Edward David Moreno, Edson Horta, Eduardo Lima, Durgam Vahia

Abstract

This book chapter presents the contribution to background of researcher and professionals in computer architecture and topics related, because we propose the deeply discussion and the curricula describing many classes in computer architecture. The idea is based on replying to the question: what kind of academic perfil and professional must adopt to former the students in new era of new technologies & processors ? How the best curricula to implement the catalog of courses in computer architecture area? The main goal of this book chapter is focused in describing the syllabus of courses to provide the support and background in computer architecture.

Resumo

Este capítulo de livro apresenta uma contribuição para formação de pesquisadores e profissionais na área de Arquitetura de Computadores e áreas afins, uma vez que, discute e propõe um elenco de disciplinas avançadas de pós-graduação concebendo a idéia de futuro em Arquitetura de Computadores. A idéia é respondermos à pergunta: que tipo de perfil acadêmico/profissional devemos formar diante das novas e futuras tecnologias de processadores? Qual seria a melhor grade curricular para este elenco de disciplinas que dê suporte a formação em Arquitetura de Computadores e áreas afins? O objetivo é apresentar uma ementa resumida do elenco das disciplinas propostas e introduzir com mais detalhes a disciplina de Arquiteturas Avançadas de Computadores I. Por fim, discutiremos as experiências com uso de simuladores, trabalhos futuros e o foco das atividades em laboratório

5.1. Introdução

Este capítulo tem como objetivo principal apresentar um elenco de disciplinas visando a formação de um pesquisador acadêmico e de um profissional especialista na área de Arquitetura de Computadores, com foco na nova era de tecnologias emergentes impulsionadas pelas barreiras de memória, potência e largura de banda.

A inovação neste capítulo é apresentar um currículo diferenciado composto por um núcleo de quatro disciplinas, sendo duas disciplinas base já existentes (Arquitetura Avançadas de Computadores e Projeto de Sistemas Reconfiguráveis) e duas disciplinas propostas com o intuito de serem implementadas em um período de dois anos (Projeto de Microprocessadores Especiais e Programação em Computação de Alto Desempenho - HPC).

Esse núcleo de disciplinas é coordenado pelo grupo de Computação Pervasiva e Alto Desempenho (PAD/LSI USP) e se relaciona com outras disciplinas existentes no Departamento de Sistemas Eletrônicos da Escola Politécnica da USP, proporcionando duas linhas de especialização, uma em desenvolvimento de aplicações HPC e outra em projeto de microprocessadores.

As disciplinas mais diretamente relacionadas são: Jogos Eletrônicos Interativos, Processamento e Análise de Imagens e Vídeos e Projeto de Sistemas sobre Silício – Tarefas “System-Level”. Essa estrutura é apresentada na Figura 01.

Essas disciplinas terão como abordagem proporcionar meios do aluno efetivamente compreender aspectos como a microarquitetura de processadores através de ferramentas como simuladores, emuladores em FPGA e outros, bem como oferecer oportunidade de compreender o impacto da mudança de parâmetros e características da microarquitetura no desempenho, custo e consumo de energia e programabilidade.

Pretende-se oferecer neste currículo uma base sólida em Arquitetura de Computadores, conhecimento das técnicas de reconfiguração parcial de FPGA e Programação para Computação de Alto Desempenho e Sistemas Heterogêneos.

As linhas de pesquisas envolvidas são:

- Desenvolvimento de aplicações de processamento de imagens e vídeo (especialmente simulação de processamento de sinais UWB);
- Desenvolvimento de jogos;
- Projeto de processadores especiais, utilizando como base o projeto de chip do OpenSPARC T1/T2, para o desenvolvimento de pesquisas relacionadas com integração SoC (System on Chip), interface de rede, PCIe, controladores DRAM e memória cache.

O aluno que iniciar o currículo fará as disciplinas de acordo com seu plano de pesquisa e planejamento da estrutura curricular a ser discutida com o orientador. A quantidade de disciplinas é estipulada pelo programa de pós-graduação em Engenharia Elétrica da POLI/USP.

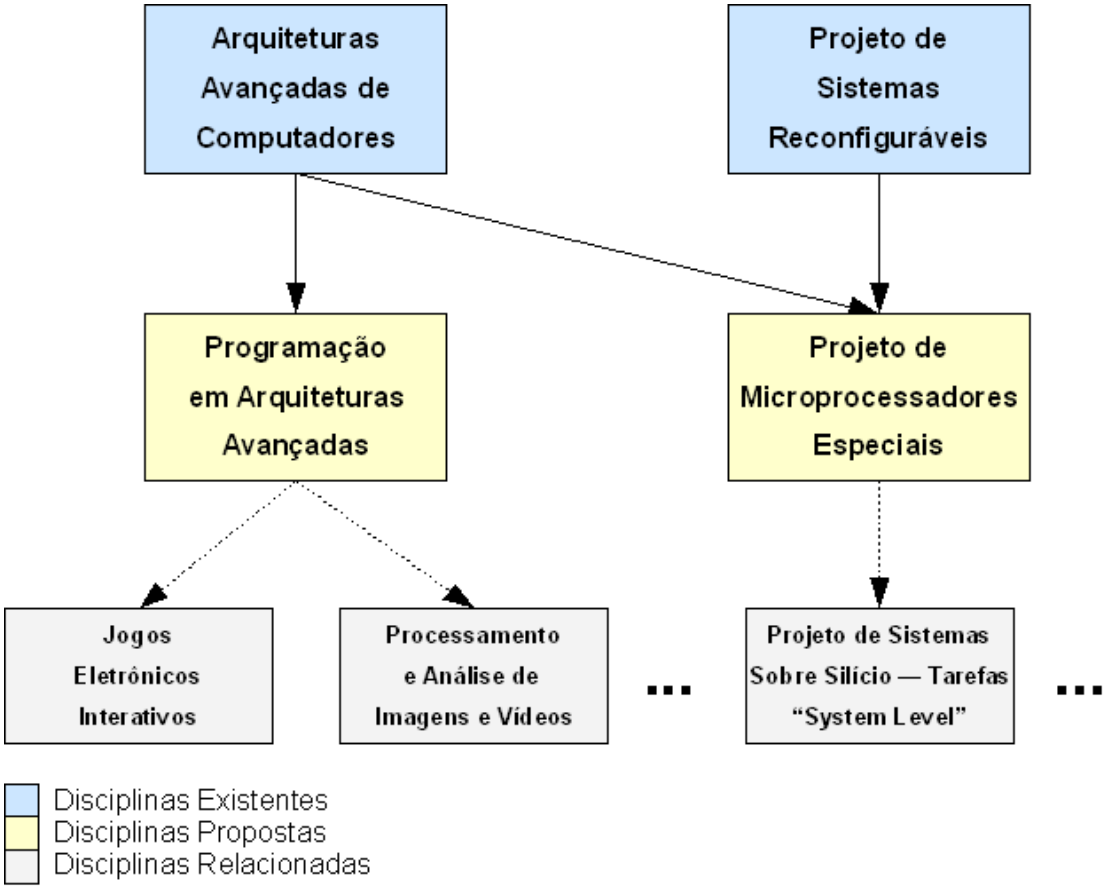


Figure 5.1. Currículo de Engenheiro de Hardware e HPC

Analisando esta proposta de Elenco de Disciplinas para formação de currículo, pretendemos responder a seguinte questão: que tipo de perfil acadêmico/profissional poderemos formar no nosso programa em especial? E como agregar as pesquisas já consolidadas no departamento para dar suporte às disciplinas que serão propostas.

Segundo David Patterson [Patterson 2008], em sua última palestra sobre o "Futuro da Arquitetura dos Computadores", há duas classes interessantes de computadores para o futuro:

- Datacenter é um computador. Amazon, Google, Microsoft constroem 50.000 mil computadores para rodar software como um serviço (Software as a Service – SaaS).
- Celular é um computador. Milhões de celulares a cada dia estão promovendo um incremento cada vez maior de funcionalidade.

A questão mais importante abordada por David Patterson é a descrição das áreas que trarão uma grande oportunidade para a Arquitetura de Computadores, comentadas a seguir:

- Tendências Manycore: o objetivo da área é inventar computadores capazes de executar os programas de forma eficiente, portável, correta e escalável. Esta tendência permite que o software use muitos co-processadores

simples com energia eficiente ao invés de um único processador que dispenda uma grande quantidade de energia.

- Nova arquitetura para Segurança: propõe nova arquitetura apoiada em itens de segurança e privacidade enfrentados na área de tecnologia da informação (TI).Novas características estão sendo estudadas em um projeto de chip para acelerar a construção, tornar seguro e diminuir o overhead das máquinas virtuais de modo que o software possa migrar entre o datacenter e o celular.
- Arquiteturas com suporte a programação altamente produtiva: propõe novos computadores que irão remover os gargalos de desempenho para um novo ambiente de programação altamente eficiente tal como o Ruby ou Python. Estas linguagens permitem uma redução do número de linhas de código de forma considerável. Então, inventar computadores que suportem tais sistemas (Ruby e Python) pode aumentar a escala de processadores preservando a produtividade.

Neste capítulo discutiremos a atualização e descrição das ementas das disciplinas, o inter-relacionamento, experiências didáticas e resultados das disciplinas existentes e a proposta de duas novas disciplinas para o programa.

Como objetivo específico do capítulo, temos: apresentar uma proposta de elenco de disciplinas para formação de um currículo; proposta de disciplinas (elaboração de ementas atualizadas) que demandam HPC e projeto de processadores; uma discussão detalhada das ementas (disciplinas existentes) e seus inter relacionamentos; e concluir a importância e contribuição deste trabalho.

Este capítulo está organizado nos seguintes tópicos: Introdução, Abordagens de Ensino e Contexto no Brasil, Proposta do Elenco de Disciplinas, Proposta de Disciplinas, Disciplinas Existentes, Inter-relacionamento de Ementas e Conclusão.

5.2. Abordagens de Ensino e Contexto no Brasil

Não há na literatura uma abordagem totalmente definida a respeito desta questão de ensino em Arquitetura de Computadores. Contudo, temos de nos conscientizar que certos enfoques de estrutura de hardware e projeto de chip não são tão relevantes em determinados currículos de ciência da computação, mas sim mais pertinentes para engenharia da computação [Clements 2009]. Futuras direções do currículo em Arquitetura são discutidas em um artigo sobre a utilização compilação em silício para hardware/software co-design [Downton 2002], uma contribuição da Universidade de Essex, Reino Unido.
Técnicas de aprendizagem [Kock 2003] são abordadas por Kock e referenciadas em nossas disciplinas propostas.

A elaboração desta proposta visa contemplar também as necessidades e impactos na sociedade em que alunos e professores estão inseridos, a realidade do mercado de trabalho, bem como a capacidade de provocar mudanças nesses contextos. Visa também

a atualidade do conteúdo técnico (estado da arte) e a busca por novos horizontes tecnológicos [Masetto 2003].

As tendências multicore e manycore fazem com que seja introduzido no elenco de disciplinas algo que já vem sendo explorado na disciplina atual de Arquiteturas Avançadas de Computadores I: o surgimento dos novos Centro de Competência IBM Cell [Portal CellBR 2007] e Centro de Excelência (CoE) SUN em OpenSPARC [OpenSPARC Workshop 2008, OSUM OpenSPARC Brazil 2008]. Tais centros promovem contribuições em recursos materiais e ferramentas para o desenvolvimento de um bom currículo de processadores avançados no ensino de Arquitetura de Computadores.

5.3. Proposta de Elenco de Disciplinas

Apresentaremos o elenco das disciplinas relacionadas com a formação em Arquitetura de Computadores no contexto da Pós-Graduação em Engenharia Elétrica da Universidade de São Paulo – USP.

Nesta seção, discutiremos a estrutura do elenco das disciplinas e focaremos inicialmente na base do currículo em Arquitetura de Computadores, que são as disciplinas já existentes no departamento.

Figure 5.2. Disciplinas Existentes

No momento, as disciplinas que já existem no Programa de Engenharia Elétrica são: Arquiteturas Avançadas de Computadores I e Projeto de Sistemas Reconfiguráveis, que estão em processo de revisão de ementa.

Como estas disciplinas são a base para o nosso elenco de disciplinas, iniciaremos com a apresentação das disciplinas, ementa, e experiências.

5.3.1 Arquiteturas Avançadas de Computadores

Atualmente, esta disciplina é realizada em 12 semanas, composta de tópicos avançados em Arquitetura de Computadores, de modo que aluno tenha uma visão do que há de mais avançado em termos de organização de sistemas híbridos (CPU convencional, GPU, FPGA, CELL), e principalmente arquiteturas paralelas.

No sistema FENIX da USP, existem todas as disciplinas oferecidas por escolas. No momento, a disciplina de Arquiteturas Avançadas de Computadores está sendo atualizado para atender a nova Era *Exascale*.

Para cada disciplina, o sistema apresenta as seguintes informações: Código, Nome, Área de Concentração, Data de Criação, Número de Créditos, Carga Horária – Total em horas/semanas, Teórica, Prática, Estudos, Objetivos, Justificativa, Ementa, Metodologia, Forma de Avaliação e Bibliografia.

A seguir, apresentamos a ementa da disciplina Arquitetura de Computadores:

Disciplina PSI5846
Arquiteturas Avançadas de Computador I

Área de Concentração: 3142

Criação: 30/06/2003

Ativação: 30/06/2003 **Desativação:**

Nr. de Créditos: 8

Carga Horária

Teórica (por semana)	Prática (por semana)	Estudos (por semana)	Duração	Total
3	0	7	12 semanas	120 horas

Docentes Responsáveis:

Sergio Takeo Kofuji

Márcio Lobo

Importância da Disciplina:

Introduzir Arquiteturas Avançadas de Computadores com ênfase em processadores especiais como aceleradores de aplicações científicas e de engenharia.

Questionar com o aluno a questão do desempenho computacional em função da carga de trabalho e do custo. Introduzir questões de pesquisa em alta como a programabilidade para arquitetura multicore.

Conteúdo

1.Revisão de Arquitetura de Computadores
Introdução de conceitos
Métricas de Desempenho Computacional
Tecnologias
Geração de Computadores
Pipeline de Processadores
Tendências em Escalabilidades
Classes de computadores

2.Microarquiteturas Avançadas
Arquitetura RISC e CISC
Arquiteturas Superescalares
Arquiteturas VLIW

3.Classificação de Arquiteturas Paralelas (Flynn)
Arquitetura SMP
Arquitetura SMP/NUMA
Arquitetura SMP/NUMA Híbrida

4.Arquiteturas Multicore
Arquitetura de Memória Compartilhada
Arquitetura de Memória Distribuída

5.Arquitetura Multithreading
Limitações no Projeto de Chip
Hardware CMP, CMT, HMT
Gerações SPARC
Microarquitetura do SPARC V8/V9
Virtualização de Hardware
Introdução ao processador OpenSPARC T1/T2

6.Arquitetura Heterogênea
Introdução ao Ecosistema Cell BE
Introdução ao Processador Cell Broadband Engine
Visão geral do Cell Hardware (Componentes: PPE e SPE's)
Visão geral do Cell Software (SDK, ferramentas, simulador e bibliotecas otimizadas)
Arquitetura Cell BE
Conceitos básicos de programação para Cell (DMA, Mailboxes, SIMD)
Introdução ao Cell SDK 3.1
Hands-on (Programa Hello World PPExSPE)
Hands-on (Programa DMA e Mailboxes)
Hands-on (Programa SIMD)
Hands-on (Ferramenta para análise estatística do tempo da SPU)

7.Arquitetura Many-core
Introdução à processadores many-core, processadores multicore e programação C (VSCSE Summer School, Wen-mei W. Hwu and Nady Obeid, 2009).
Modelo de Programação CUDA (Lecture 2, David Kirk/NVIDIA and Wen-mei W.Hwu, 2009).
CUDA Threads (Lecture 3, David Kirk/NVIDIA and Wen-mei W.Hwu, 2009).
CUDA Memories (Lecture 4, David Kirk/NVIDIA and Wen-mei W.Hwu, 2009).

8.Processadores
Processadores Comerciais
Servidores Blades
Servidor Blade Cell
Servidor Blade Cell Híbrido (Cell + GPU)
Cluster de FPGA (Xilinx)

9.Tópicos Especiais
Sistemas Embarcados
Introdução à FPGA

Ementa:
Revisão de Arquitetura de Computadores. Microarquiteturas Avançadas. Classificação das Arquiteturas Paralelas (Flynn). Arquiteturas Multicore. Arquiteturas Multithreading. Arquitetura Heterogênea. Arquitetura Many-core. Processadores. Tópicos Especiais.

Bibliografia
H. El-Rewini, M. Abd-El-Barr, "Advanced Computer Architecture and Parallel Processing, Wiley, 2005. OLUKOTUM, Kunle. Chip Multiprocessor Architecture.Synthesis Lectures on Computer, 2007. David Weaver. OpenSPARC Internals, 2008. Architecture, Morgan.Culler et al. Parallel Computer Architecture: A H/S Approach. 1999. PATTERSON, David & Hennessy. Computer Architecture: Quantitative Approach**,** 3rd Edition, 2007. PATTERSON, David & Hennessy. Computer Organization and Design: the Hardware /Software Interface**,** 3rd Edition, Elsevier, 2004. ASONOVIC, Krste et al. The Landscape of Parallel Computing Research: A View from Berkeley**.** Relatório Técnico No UCB/EECS-2006-183, 2006. Disponível em:
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

O objetivo geral da presente proposta de disciplina de pós-graduação em Arquiteturas Avançadas de Computadores é apresentar um currículo complexo que aborda desde uma revisão dos conceitos de Arquitetura de Computadores até uma visão do sistema de memória e o *pipeline* de processadores avançados através de simuladores, sistemas de hierarquia de memória, pipeline complexo, taxonomia de arquiteturas paralelas, arquiteturas *multithread*, arquiteturas *multicore.* Como arquiteturas de referncia, serão adotados o OpenSparc T1 e T2 (multicore homogêneo como suporte a multi-thread simultâneo - SMT), CELL BE (multicore heterogêneo), GPUs (*manycore*) e FPGAs (Aceleradores de Aplicações).

Introduziremos as disciplinas: Arquiteturas Avançadas de Computadores e Projeto de Sistemas Reconfiguráveis como pré-requisitos para realizar nosso elenco de disciplinas propostas. A revisão citada no primeiro tópico do conteúdo, trata abordagens apresentadas em um curso introdutório de arquitetura de computadores segundo [Loui 1988, Fishwick 2000, Simeonov 1995, Yehezkel 2002]. A disciplina de Arquiteturas Avançadas de Computadores I propõe o uso de dois principais simuladores: *Simics* [Virtutech], e o *IBM full System Simulator for Cell Broadband Engine* [IBM Simulator]. Para o uso do simulador "Simics", analisaremos o sistema de memória e os protocolos de cache dos seguintes processadores: Toshiba Cell PPU/Toshiba Cell SPU e UltraSPARC T1/UltraSPARCT2. Existem outros trabalhos correlatos consagrados, como o simulador *Multifacet GEMS* [Martin 2005] pela Universidade de Wisconsin, específico para arquiteturas SMP e simulam eficientemente arquiteturas CMP.

Para a disciplina proposta de Microprocessadores Especiais, se fará uso da ferramenta para projeto de chip, o RTL Verilog do OpenSPARC T1/T2 [OpenSPARC].

A disciplina faz referência a três cursos de Arquiteturas Avançadas de Universidades de excelência e pesquisadores renomados como: UC Berkeley [Asanovic], Instituição Politécnica de Budapeste [Sima] e Universidade SMU [El-Rewini]. Para revisão dos conceitos de arquitetura e organização de computadores adota-se o livro "Organização e Projeto de Computadores: Interface Hardware/Software", além do "Arquitetura de Computadores: Uma abordagem Quantitativa" [Patterson, Hennessy]. Como bibliografias básicas para o contexto do curso são adotados os seguintes livros: para aprendizagem de programação Cell, o livro: "Programming the Cell processor: For games, graphics and computation" por Mattew Scarpino [Scarpino 2008] e para aprendizagem de arquitetura multithread e OpenSPARC, os livros:

Chip Multiprocessor Architecture: Techniques To Improve Throughput and Latency, por Kunle Olukotum *et al*. [Olukotum 2007];

OpenSPARC Internals por David Weaver [Weaver 2008].

5.3.1.2. Simuladores para Ensino e Pesquisa em Arquitetura

Nesta seção, o objetivo não é descrever detalhes dos simuladores bem como abordar o seu uso dentro de uma perspectiva de pesquisa, mas comentar brevemente a importância destes simuladores em especial para a disciplina de Arquitetura de Computadores. No nosso contexto, tanto o Simics quanto o IBM full-system Simulator já estavam sendo estudados na disciplina de Arquitetura Avançada de Computadores. Apenas o foco de pesquisa relacionado com o simulador Simics reestringiu-se no momento aos processadores OpenSPARC e CELL. Para a ferramenta OpenSPARC temos interesse específico para projeto de chip e não o uso do simulador.

Por isso, a intenção é propor sugestões de uso de simuladores para a comunidade brasileira no que diz respeito a disciplinas de Arquitetura de Computadores e outras relacionadas.

5.3.1.3 Simics (utilizado na disciplina de Arquitetura Avançadas de Computadores I)

Por que devemos utilizar o Simics em Laboratório? Segundo Magnusson [Magnusson, 2002] o ambiente de software do Simics nos permite as seguintes atividades: rodar um código em diversas plataformas (ISA's), testar benchmarks e experimentos, modificar características no hardware (através do acesso de múltiplas camadas de abstração) e finalmente analisar como os mecanismos arquiteturais trabalham na prática com um software real.

Utilizamos o Simics para uma análise do programa, pesquisas em arquitetura de computadores, e debugg do Kernel [Scott Beamer 2009].

A presente pesquisa está restrita em modificações nas características arquiteturais do processador SPARC e no processador CELL.

5.3.1.4 IBM-full System Simulator for Cell Broadband Engine (utilizado na disciplina de Arquitetura Avançadas de Computadores I)

O propósito do uso do "*IBM full-System Simulator for Cell Broadband Engine*" é possibilitar que os desenvolvedores de aplicação tenham contato com a nova tecnologia, sem a necessidade de se ter o hardware real disponível, auxiliando até em uma simulação prévia, e tornando as aplicações mais eficientes quando forem executadas em uma máquina real, seja em uma IBM Blade Cell, Sony PS3 ou Toshiba Cell.

Para o laboratório prático tem como objetivos: iniciar o simulador no modo texto e no modo gráfico; como utilizá-lo para rodar o Linux sobre o processador Cell BE no ambiente do simulador; como utilizar a funcionalidade de profile do simulador para obter dados sobre o desempenho da execução de uma determinada aplicação.

Outro enfoque que será dado no laboratório com uso do Simulador Cell é o OpenCV no Cell. Apresentaremos um exemplo prático de um aplicativo portado da plataforma x86 para a plataforma Cell BE. O objetivo geral é analisar algoritmos simples de processamento de imagens (subdivisão das imagens em partes menores para processamento e tratamento de bordas de regiões). Como objetivo específico, apresentaremos a biblioteca OpenCV integrada no "IBM full-System Simulator", suas capacidades e funcionalidades.

Além disso, este exemplo servirá de pano de fundo para o desenvolvimento de conceitos específicos relacionados à programação para a arquitetura Cell como, por exemplo, o uso de DMAs para transferências de dados entre os diversos núcleos e de double e multi-buffering para aumentar a eficiência destas transferências.

A justificativa para este enfoque é estudar o processador através de aplicações específicas para processamento de imagens e também dar suporte à disciplina de "Algoritmos para Processamento e Análise de Imagens" do Departamento de Sistemas Eletrônicos. Os resultados desta pesquisa foram apresentados em um concurso da IBM que ganhou o prêmio de 4º lugar nas Américas: "*Cell Contest: Beyond on Gaming 2007*", e em uma Tese de Doutoramento intitulada [Hiramtasu 2008]: "*Sistema de Visão Computacional sobre Processadores com Arquitetura de Múltiplos Núcleos*".

5.3.1.5 Ferramentas para OpenSPARC I

Optamos por utilizar o processador OpenSPARC T1, porque ele possibilita o estudo de implementação de FPGA no processador OpenSPARC T1. Foi adquirido uma placa da Xilinx ML401 com alteração do chip. Para implementar dois cores do OpenSPARC em um FPGA (Virtex 5 de 65nm) é necessário uma placa específica com um custo bem elevado. Atualmente, a comunidade OpenSPARC já conta com um kit da Xilinx ML505 FPGA (Virtex 5 XUPV5-LX110T). Como o foco do Centro de Excelência em OpenSPARC é a implementação do FPGA no OpenSPARC, então o nosso objeto de estudo é o OpenSPARC T1.

Na disciplina de Arquiteturas Avançadas de Computadores, introduzimos apenas a parte teórica da microarquitetura do OpenSPARC, abordando um padrão de simulação muito adotado no mercado e na academia, o Simics.

Atualmente, a disciplina de Projeto de Sistemas Reconfiguráveis, introduz dispositivos lógicos programáveis e apresenta as técnicas de reconfiguração parcial, implementando o projeto em uma placa bem mais simples, sem introduzir a questão de uma arquitetura multicore em um FPGA.

O OpenSPARC T1 possui dois pacotes importantes: um para projeto de processador (OpenSPARC T1 Chip Design) e outro para estudo da Arquitetura e Modelagem de Performance (OpenSPARC T1 for Architecture and Performance Modeling Tools).

O Verilog RTL para OpenSPARC T1 é um pacote para simulação de projeto e verificação do chip OpenSPARC T1 baseado no processador UltraSPARC V8. Este pacote faz parte da Comunidade OpenSPARC que disponibiliza o código fonte dos processadores: UltraSPARC V8 (32 bits) e UltraSPARC V9 (64 bits) [Mehta 2006].

Na disciplina que está sendo proposta: Projeto de Microprocessadores Especiais, é que vamos introduzir o estudo aprofundado da microarquitetura do OpenSPARC T1, realizando a simulação do projeto inicialmente. O objetivo maior é criar um currículo de referência para se trabalhar com o ambiente do "OpenSPARC T1 Chip Design". Os temas a serem abordados serão: coerência de memória e a comunicação dos processadores com os periféricos. Atualmente, o pacote OpenSPARC T1 1.6 Release já implementa 4 threads no T1 core para o Virtex 5 FPGAs:

- Placa ML505-V5LX110T
- Arquivos de projetos EDK (EDK 9.2)
- Scripts para rodar a regressão RTL em hardware

Em Berkeley, o projeto RAMP [Thatcher 2009], já está testando a implementação do FPGA em OpenSPARC para projeto multicore.

Inicialmente, será proposto na disciplina os seguintes objetivos: ensinar como customizar e usar o OpenSPARC, rodar um código OpenSPARC, modificar características básicas (configuração do número de cores e threads) e finalmente configurar o ambiente de simulação.

A arquitetura OpenSPARC T1 possui simuladores para o modelo de arquitetura SPARC. Podemos citar: SAM e o Legion.

Introduziremos o SAM para análise de desempenho, geração de traces, e simulação dirigida à execução. Já o Legion, é um simulador que possui um ambiente

para simulações rápidas, desenvolvendo e testando funcionalidade de software em hardware.

5.3.1.6 Projeto de Sistemas Reconfiguráveis

Disciplina PSI5003
Projeto de Sistemas Reconfiguráveis

Área de Concentração: 3142

Criação: 07/10/2004

Ativação: 07/10/2004 **Desativação:**

Nr. de Créditos: 8

Carga Horária

Teórica (por semana)	Prática (por semana)	Estudos (por semana)	Duração	Total
3	0	7	12 semanas	120 horas

Docentes Responsáveis:

Sergio Takeo Kofuji

Edson Lemos Horta

Conteúdo

Dispositivos Lógicos Programáveis (EPLDs, CPLDs, FPGAs); reconfiguração em tempo de compilação; reconfiguração em tempo de execução; reconfiguração parcial em tempo de execução; metodologia de projeto de sistemas parcialmente reconfiguráveis; ferramentas de projeto; linguagens de descrição de hardware (VHDL e Verilog); simulação de sistemas reconfiguráveis; estudo de casos de sistemas parcialmente reconfiguráveis; projeto de um sistema parcialmente reconfigurável.

Bibliografia

HDL Chip Design: A Practical Guide for Designg, Synthesizing and Simulating ASICs and FPGAs Using VHDL or Verilog Douglas J.Smith, Foreword by Alex Zamfirescu - Doone Publications - Março 1998; - Writing Testbenches: Functional Verification of HDL Models, Janick Bergeron - Kluwer Academic Publishers - 2000 - COMPTON, K. Configurable computing: A survey of systems and software. Northwestern University, Dept. of ECE, 1999. Relatório técnico. - HUTCHINGS, B.L. Implementation approaches for reconfigurable logic applications, In: Moore, W., editors, FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, 1995. Springer-Verlag, Berlin, 1995. p.419-428. - GUCCIONE, S.A. Design advantages of run-time reconfiguration, In: Schewel, J. et al., editors, RECONFIGURABLE TECHNOLOGY: FPGAS FOR COMPUTING AND APPLICATIONS, PROC. SPI 3844, 1999. Anais, Bellingham, WA:SPIE (The International Society for Optical Enginnering), 1999. p.87-92.

Nesta disciplina, os alunos demonstraram um interesse muito grande na utilização de dispositivos lógicos programáveis para implementar os projetos de pesquisa relacionados com suas respectivas dissertações e teses. Através dos laboratórios propostos, foi possível aplicar a metodologia de projeto com o auxílio de ferramentas de desenvolvimento atuais. Os exemplos práticos foram muito úteis na fixação dos conceitos apresentados nas introduções teóricas. A verificação de novas arquiteturas foi realizada através de simulações funcional e de tempo, após a implementação das mesmas no dispositivo lógico programável (FPGA).

5.4 Proposta de Disciplinas

Uma das maiores contribuições deste capítulo é a proposta de duas disciplinas importantes para o currículo de Arquitetura de Computadores da Escola Politécnica. Estas duas disciplinas que oferecem duas oportunidades de pesquisas bem importantes para esta nova era “Exascale” que estamos entrando:

- Projeto de Microprocessadores Especiais: Disciplina baseada em microprocessadores CMT (Chip Multithread).
- Programação em Arquiteturas Avançadas: Disciplina baseada em tópicos avançados de programação para Computação de Alto Desempenho em processadores aceleradores (GPU e FPGA).

5.4.1 Projeto de Microprocessadores Especiais

A disciplina de Projeto de Microprocessadores especiais visa oferecer ao aluno uma introdução ao projeto de microprocessadores CMT baseado em código aberto. O foco da disciplina é introduzir o ambiente de projeto de microprocessadores multicore, utilizando o Verilog RTL para o OpenSPARC T1.

Espera-se que ao final do curso, o aluno esteja familiarizado com vários tópicos quentes de pesquisa como integração SoC, rede de interconexão (*inter-connection network*), comunicação de dados entre processadores e memória, controlador de memória DRAM e cache.

O curso é baseado em aulas teóricas com tópicos introdutórios e de pesquisa e finalmente alguns laboratórios.

Conteúdo:

1. Introdução ao Chip Multithreading (CMT)
Processadores SMT (Simultaneous Multithreading, IEEE Micro 1997)
Recentes tendências em arquitetura de processadores (NIT Treaty India, 2007)

2. Introdução ao OpenSPARC T1
Tutorial ASPLOS XII (Innovating with OpenSPARC)
Microarquitetura do OpenSPARC T1

3. Tópicos de Pesquisas em OpenSPARC
Tópicos de Pesquisa em OpenSPARC (OpenSPARC: An Open Platform for Hardware Reliability Experimentation, SELSE 2008)
Implementação do FPGA em OpenSPARC (Multicore Expo, Tokyo 2007)

4. Projeto de Chip OpenSPARC
Requisitos do Sistema e Ferramentas EDA
Como customizar e usar o OpenSPARC (OpenSPARC Internals)
Hands-on 1: Execução dos pacotes “test bench environments” (Chip 1, Chip8)
Hands-on 2: Rodar a primeira síntese

5. Síntese do Projeto no OpenSPARC
Sintetizando o projeto OpenSPARC usando a biblioteca com tecnologia Synopsys 90nm
Sintetizando o projeto OpenSPARC usando a biblioteca aberta Nangate 45nm

6. Conceitos importantes da Microarquitetura
Introdução à Integração SoC (System-on Chip)
Visão Geral do CPU-cache-crossbar (CCX)
Coerência de memória – L2 cache e Ordenamento de instrução
Controlador DRAM
PCIe

7. Configuração do Ambiente de Simulação
Simulação e Emulação do Ambiente

8. Verificação do Projeto OpenSPARC

Ementa:

Introdução à Chip Multithreading. Introdução ao OpenSPARC T1. Tópicos de Pesquisas em OpenSPARC. Projeto de Chip OpenSPARC. Síntese do Projeto no OpenSPARC. Conceitos importantes na Microarquitetura. Configuração do Ambiente de Simulação. Verificação do Projeto OpenSPARC.

Bibliografia

OLUKOTUM, Kunle**.** Chip Multiprocessor Architecture. Synthesis Lectures on Computer, 2007. David Weaver (2008). "OpenSPARC Internals". OpenSPARC T1 Microarchitecture Specification, Sun report 2008. OpenSPARC T1 processor design and verification – User's Guide, Sun report 2008.

5.4.2 Programação em Arquiteturas Avançadas

A proposta desta disciplina pretende inspirar o desenvolvimento de um novo curso que aborde aplicações de dados intensivos em larga escala, propondo uma base para simulações científicas de engenharia em sistemas heterogêneos e no futuro, em sistemas híbridos com cluster de Cell + GPU + FPGA.
Atualmente, a computação escalável é de extrema importância para as áreas de ciência e engenharia, devo a questão do tempo despendido com grandes modelos matemáticos.

Um dos objetivos ao oferecer este curso é preparar os estudantes da engenharia com interesse em processamento de imagens, eletromagnetismo, introduzindo conceitos elementares de algoritmos da computação, imagem e física como estudo de caso para realização de atividades práticas de simulações.

Nesta disciplina, pretendemos oferecer um breve introdução em programação paralela (MPI, OpenMP) e técnicas de debugg paralelo. Ao apresentar estas técnicas de debugg pretende-se que o aluno tenha um noção de um programa seqüencial, MPI, OpenMP; escrito em C e Fortran.

Ao final do embasamento teórico, espera-se que o aluno esteja familiarizado ambiente computacional de alto desempenho e compiladores paralelos para as seguintes plataformas: IBM Blade Cell QS21, SUN Fire 10000.

Estaremos introduzindo o desenvolvimento de aplicações com o paradigma da programação CELL. A vantagem da utilização deste paradigma é a interface com usuário em termos de ambiente de desenvolvimento integrado com IBM full System Simulator para o processador CELL. O simulador dispensa qualquer plataforma, mas como a intenção é obter ganho de desempenho nas aplicações há suporte para simulação em plataforma real (IBM Blade Cell ou PS3).

Conteúdo:

1. Introdução aos Sistemas de Alto Desempenho do PAD

Visão Geral do Sistema Kyoto (IBM Blade Cell QS 21) Visão Geral do Sistema Danger (SUN Fire 10000)

Requisitos de Software

2. Introdução à Programação Paralela

Introdução à Programação MPI

Introdução à Programação OpenMP

Laboratório MPI (Hands-on)

Laboratório OpenMP (Hands-on)

Técnicas de Compilação e Otimização

3. Laboratório de Fundamento sobre Equação de Jacobi

Introdução sobre Equação de Jacobi

Exercício 1: Entender a sintax do algoritmo C/C++ e compilar na arquitetura HPC.

Exercício 2: Compilar o algoritmo OpenMP do Jacobi

Exercício 3: Paralelização de Memória Distribuída (Algoritmo MPI do Jacobi)

4. Programação CELL

Introdução ao uso do Simulador para o processador CELL

O uso da biblioteca OpenCV na Blade Cell QS21

Algoritmos para Processamento de Imagens

Algoritmos para Aplicações UWB

Introdução ao Algoritmo FDTD para CELL

Bibliografia:

Maurice Herlihy, Nir Shavit. "Art of Multiprocessor Programming". Elsevier, 2008. Barbara Chapman, Gabriele Jost, Ruud van der Pas. "Using OpenMP - Portable Shared Memory Parallel Programming". Darryl Gove. "Solaris Application Programming", 1st version, Prentice Hall, 2008. David E. Culler and Jaswinder Pal Singh, with Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* 1998, ISBN: 1558603433. Michael J. Quinn. *Parallel Programming in C with MPI and Open MP*. McGraw Hill. 2003. ISGN: 0072822562.

5.5 Conclusão

A proposta inovadora do elenco de disciplinas apresentado neste capítulo, com algumas já sendo ministradas na Escola Politécnica da Universidade de São Paulo – USP, está ainda em uma fase inicial, de discussão com os vários professores do departamento. A hierarquia de disciplinas apresentada, bem como o inter-relacionamento entre elas, ainda deverão passar por um período de análise para se tornar um currículo de referência.

Neste capítulo, foram definidas as disciplinas base para o currículo: Arquiteturas Avançadas de Computadores e Projeto de Sistemas Reconfiguráveis. A disciplina Arquiteturas Avançadas de Computadores foi totalmente atualizada, mantendo o seu propósito original de apresentar tópicos avançados em arquitetura de computadores, porém, agora consolidando o laboratório prático do uso de simuladores para o mais variados fins.

A disciplina de Projeto de Sistemas Reconfiguráveis vem trazendo uma contribuição importante no que refere à realização de projeto prático, abordando as técnicas de reconfiguração parcial, implementadas em um FPGA da Xilinx. A contribuição que vamos propor na disciplina será a inserção de uma arquitetura multicore, mais especificamente um chip multithreading. Adotaremos o uso do OpenSPARC, por possuir o código fonte do chip aberto, ideal para disciplina, porque poderemos testar a implementação de dois cores em um FPGA.

A proposta de mais duas disciplinas para compor o currículo de pós-graduação do departamento de sistemas eletrônicos surgiu do amadurecimento das pesquisas desenvolvidas no Centro de Competência Cell da IBM e o Centro de Excelência em OpenSPARC da SUN na USP. São dois novos centros de grande importância em termos de tecnologia CELL e tecnologia SPARC, respectivamente.

A disciplina de Projeto de Microprocessadores Especiais introduz o currículo OpenSPARC, uma vez que aprofunda na arquitetura CMT e utiliza o RTL Verilog do OpenSPARC T1 para abordar tópicos extremamente relevantes no meio científico e no mercado de projeto de processadores como integração do SoC, interface de rede, PCIe e controladores de memória DRAM.

Já a disciplina Programação Avançada de Computadores, propõe um aprendizado único em computação de alto desempenho escalável para simulações científicas na área da ciência e engenharia. No futuro serão realizadas simulações em processadores especiais como GPU e FPGA, incorporados na arquitetura da IBM Blade Cell. A disciplina traz embasamento teórico e prático em programação paralela e programação cell, introduz técnicas de compilação e otimização, bibliotecas, algoritmos para processamento de imagens e aplicação UWB.

De acordo com a proposta do elenco de disciplinas, como trabalho futuro, pretendemos descrever mais três disciplinas importantes para o currículo já existentes no departamento de sistemas eletrônicos: Jogos Eletrônicos Interativos, Processamento e Análise de Imagens e Vídeos e Projetos de Sistemas sobre Silício.

Referências

Patterson, David (2008). “Future of Computer Architecture”, Talks at UC Berkeley, December 12, 2008.

Downton, A.C., Fleury, M., Self, R.P., Noakes, P.D. “Future Directions in Computer Architectures Curricula: Silicion Compilation for Hardware/Software co-design”, Department of Electronic Systems Engineering, University of Essex, UK, 2002.

Clements, Alan. “Work in Progress – Computer Architecture Meets Ubiquitous Computing”, ASEE/IEEE Frontiers in Education Conference, Santo Antonio, TX, 2009.

Kock, N., Aiken, R., Dalton, D., Elesh, D., Ranare, A., Sandas, C. “Team Teaching an Advantage Computer Fluency Course: A composite perspective”, Proceedings of Informations Science, 0377-0383, IS 2003.

Masetto, Marcos Tarciso. Competência Pedagógica do Professor Universitário. Cap. 6 - O docente do ensino superior e o currículo do seu curso. pp.67 a 69. Summus Editorial. 2003.

Portal CellBR. Portal do Centro de Competência CELL. <www.pad.lsi.usp.br/cell>

OpenSPARC Workshop. I Workshop em Arquiteturas OpenSPARC. <www.pad.lsi.usp.br/workshop>

OSUM OpenSPARC Brazil. (2008). OpenSource University Meetup OpenSPARC Brazil Group. <http://osum.sun.com/group/opensparcbrazil>

Loui, M. C. (1988). "The case for Assembly Language", IEEE Trans. Education 31, 3, 160-164.

Fishwick, P. (2000). "Modeling the world". IEEE Potentials (March), 6-10.

Simeonovv, S., Scheinder, M. (1995). "MISM: An improved microcode simulator". ACM SIGCSE Bull.27, 2, 13-17.

Yehezkel, C., Yurcik, W., Pearson, M., Armstrong, D. (2002). "Three simulator tools for teaching computer architecture: Little Man computer, and RTLSim", ACM Journal on Educational Resources in Computing (JERIC), Vol.1, Issue 4, 60-80.

Virtutech. The Virtutech Simics Home Page. < http://www.virtutech.com/>

IBM Simulator. The IBM full-System Simulator for Cell Broadband Engine Processor Home Page. <http://www.alphaworks.ibm.com/tech/cellsystemsim>

OpenSPARC. The OpenSPARC: World's First Free 64-bit CMT Microprocessors Home Page. <http://www.opensparc.net>

Martin, M.; Sorin, D.; et al. (2005). "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset", Computer Architecture News (CAN), 1-8.

Asanovic, Krste. (2009). Computer Science 152: Computer Architecture and Engineering.

Sima, Dezo. (2009). Prof. Dr. habil Dezső Sima Site. <http://nik.bmf.hu/sima/oktatas.htm>

El-Rewini. (2008). CSE 8383: Spring 2008.

Hennessy & Patterson. (2007). Computer Architecture: A Quantitative Approach, Fourth Edition, Morgan Kaufmann.

Scarpino, Mathew. (2008). Programming the Cell Processor for Games, Graphics and Computation. Foreward by Dr. Duc Vianny (Technical Solution Architect, IBM).

Olukotun, Kunle, Hammond, Lance. (2007). Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency. Synthesis Lectures on Computer. Morgan & Claypool.

Weaver, David. (2008). OpenSPARC Internals.

Hiramatsu, K. Roberto. (2008). Sistema de Visão Computacional sobre processadores com arquitetura multi núcleos. Tese de Doutorado em Engenharia Elétrica, USP. Moscinski, Jerzy. "Changing Tools and Methods in Engineering", Proceeding of International Conference on Engineering Education – ICEE, Coimbra, Portugal, 2007.

Orgolho, E.; Falcon, A.; Faraboschi, P.; Monchiero, M.; Ortega, D. "COTSon: Infrastructure for full System Simulator", ACM Transaction on Computer Systems, Vol.43, Issue 1, 52-61, January 2009.

Magnusson, P.S.; Christensson, M.; Eskilson, J.; Forsgren, D.; Hallberg, G.; Hogberg, J.; Larsson, F.; Moestedt, A.; Werner, B. (2002). "Simics: A full-System Simulator Platform", IEEE Computer, Vol.5, Issue 2, 50-58.

Scott Beamer (2009). "Section 1: Introduction to Simics", University of Berkeley.

Shrenik Mehta, David Weaver, Jhy-Chun Wang, Ashley Saulsbury, Partha Tirumalai, Raj Prakash, "OpenSPARC T1 Tutorial", ASPLOS XII, 2006.

Thatcher, Thomas. (2009). OpenSPARC Program Updates. RAMP Retreat – January 2009, Berkeley.

Capítulo 6

Um Enfoque Interdisciplinar no Ensino de Arquitetura de Computadores

Cesar Albenes Zeferino, André Luis Alice Raabe,
Paulo Viniccius Vieira e Maicon Carlos Pereira

Abstract

This chapter presents the results of an experiment involving different courses of a Bachelor in Computer Science related to the use of a simple processors family designed to assist in learning concepts of computer architecture. The family of processors is called BIP (Basic Instruction-set Processor) and has three versions, named BIP I, BIP II and BIP III, with incremental resources and complexity. These different versions were used for teaching concepts related to Introductory Programming, Digital Circuits, Computer Architecture and Organization and Compilers. The chapter details the architecture of the BIP family of processors and discusses the results of its use in each of the courses mentioned.

Resumo

Este capítulo apresenta os resultados de um experimento envolvendo diferentes disciplinas de um curso de graduação em Ciência da Computação relacionadas ao uso de uma família de processadores simplificados, a qual foi concebida para auxiliar na aprendizagem de conceitos de arquitetura de computadores. A família de processadores foi denominada BIP (Basic Instruction-set Processor) e possui três versões, denominadas BIP I, BIP II e BIP III, com complexidades e recursos incrementais. As diferentes versões tornaram-se um recurso didático utilizado com enfoques diferentes nas disciplinas de Algoritmos e Programação, Circuitos Digitais, Arquitetura e Organização de Computadores e Compiladores. O capítulo detalha a arquitetura da família de processadores BIP e discute os resultados de sua utilização em cada uma das disciplinas mencionadas.

6.1 Introdução

A compreensão do funcionamento da arquitetura do computador e de seu processador possui importância central na formação dos alunos dos cursos de graduação em Ciência da Computação. Além de fornecer os conhecimentos básicos para possibilitar a inserção do aluno no contexto da pesquisa e do desenvolvimento de hardware, essa compreensão auxilia o entendimento da necessidade e do papel do software básico e fornece subsídios fundamentais para a aprendizagem e para a compreensão da lógica de programação.

Considerando esse último aspecto, é fato amplamente conhecido que alunos apresentam dificuldades na aprendizagem de conceitos de algoritmos e programação, em especial no primeiro ano do curso [Carbone e Kasboll 1998][Menezes e Nobre 2002][Pimentel *et al*. 2003][Good e Brna 2004][Khalife 2006]. Essa dificuldade está relacionada, entre outros aspectos, à ausência de afinidade com o raciocínio lógico formal que é o fundamento para a capacidade de abstração dos alunos. Nesse sentido, o estudo da arquitetura do computador cria a possibilidade de estabelecer relações dos conceitos de programação com aspectos concretos do hardware, reduzindo assim a necessidade de abstração.

Nesse contexto, identifica-se uma falta de sincronia nos currículos tradicionais em que o estudo da arquitetura e da organização do computador ocorre depois do ensino da programação. Para contornar esse problema, em muitos cursos, costuma-se apresentar algumas noções básicas de arquitetura e organização de computadores aos alunos em disciplinas que fornecem uma introdução geral à Computação, tipicamente no primeiro ano do curso.

No entanto, essa abordagem normalmente apresenta dois problemas: (*i*) a falta de uma articulação adequada entre os professores das disciplinas introdutórias a fim de estabelecer uma sincronia na apresentação dos conteúdos para que se possa beneficiar a aprendizagem de programação; e (*ii*) a limitação dos modelos utilizados para a apresentação dos conceitos básicos de arquitetura.

Buscando abordar esses problemas, uma pesquisa foi realizada pelos autores deste capítulo na Universidade do Vale do Itajaí (UNIVALI) a fim de desenvolver e disponibilizar uma série de processadores com um conjunto de instruções mínimo para auxiliar o aprendizado de conceitos de arquitetura e organização de computadores de modo incremental [Morandi, Raabe e Zeferino 2006][Morandi *et al*. 2006]. Dessa forma, os processadores desenvolvidos serviram de referência para a apresentação dos conceitos básicos necessários ao melhor entendimento das abstrações utilizadas nas disciplinas da área de Algoritmos e Programação.

A iniciativa de desenvolver uma família de processadores acabou permitindo que fosse dado um enfoque interdisciplinar aos conceitos de arquitetura e organização de computadores, vislumbrando seus desdobramentos em diferentes disciplinas correlatas, a saber: Computação Básica, Circuitos Digitais, Arquitetura e Organização de Computadores e Compiladores. Essa característica tornou-se um diferencial desta abordagem, uma vez que, muito frequentemente, os alunos relatam que não conseguem compreender plenamente as relações entre as disciplinas, e, nesse sentido, a abordagem utilizada fornece uma contribuição significativa.

Este capítulo busca enfatizar:

1. As diretrizes para elaboração da família de processadores proposta;
2. De que forma os conceitos foram trabalhados de maneira incremental nas diferentes disciplinas correlatas discutindo pontos positivos, negativos e dificuldades;
3. As ferramentas computacionais utilizadas e construídas para apoiar a aplicação da abordagem; e
4. Os resultados do ponto de vista didático.

Acredita-se que, por ser uma proposta que difere da abordagem tradicional de ensino na área, possua uma contribuição relevante para a comunidade interessada no ensino de Computação como um todo.

O restante deste capítulo está organizado em três seções. A Seção 2 apresenta a família de processadores BIP (Basic Instruction-Set Processor) fazendo uma análise introdutória dos processadores existentes para o ensino de Arquitetura de Computadores e fundamentando a necessidade de concepção da Família BIP. Na sequência, a seção está dividida em quatro subseções que descrevem as características gerais da Família BIP e a arquitetura e a organização de cada versão do processador. A Seção 3 detalha as estratégias, conteúdos, organização didática e ferramentas utilizadas para condução de atividades de aprendizagem com a família de processadores BIP. Apresenta-se uma descrição geral de como foi viabilizada a integração interdisciplinar e a visão dos autores sobre interdisciplinaridade em Computação. Cada uma das disciplinas envolvidas é descrita em uma subseção, a saber: (*i*) Algoritmos e Programação; (*ii*) Computação Básica; (*iii*) Circuitos Digitais; (*iv*) Arquitetura e Organização de Computadores; e (*v*) Compiladores, discutindo as formas como a arquitetura do BIP foi aplicada para apoiar o aprendizado e a consolidação de conceitos nessas disciplinas. Concluindo, a seção de considerações finais resgata os principais pontos da proposta e discute as direções futuras da pesquisa realizada.

6.2 Família de Processadores BIP

A escolha de modelos de processadores para o ensino de conceitos de arquitetura e organização de computadores é alvo de estudos frequentes pelos educadores da área, como, por exemplo, no trabalho apresentado por Clements (1999) que discute aspectos que devem ser levados em consideração na escolha de modelos de processadores a serem aplicados no ensino de graduação. Enquanto alguns autores e professores optam por utilizar modelos hipotéticos de processadores, outros adotam processadores reais e comerciais como referência para estudos de caso.

Para as fases iniciais de um curso de graduação, a seleção de processadores para o ensino concorrente da lógica de programação e de conceitos de arquitetura de computadores deve facilitar o estabelecimento de relações entre as abstrações lógicas necessárias à programação e à implementação dessas abstrações em hardware. Porém, os modelos de processadores tipicamente utilizados por professores de disciplinas introdutórias são abstratos demais e não permitem estabelecer essas relações. Uma alternativa seria utilizar modelos de processadores mais detalhados, como aqueles adotados nas disciplinas específicas da área de Arquitetura de Computadores (*e.g.* MIPS, x86,...). Porém, esses processadores são demasiadamente complexos para serem aplicados em disciplinas do primeiro ano, e poucos são os livros-texto da área

que os descrevem propiciando uma integração entre a arquitetura do processador e a programação em alto nível.

Uma exceção é o livro "Organização e Projeto de Computadores", de Patterson e Hennessy (2005), no qual os autores descrevem o processador MIPS com ênfase na exploração da interface entre o hardware e software, permitindo a interligação dos conceitos apresentados com aqueles estudados nas disciplinas de programação. Essa abordagem favorece a interdisciplinaridade e a relação entre os conceitos mais abstratos da programação de alto nível e a realização física do processador. No entanto, o pouco embasamento dos alunos nas fases iniciais torna inadequado o uso de processadores com o grau de complexidade do MIPS, e alternativas mais simples podem e devem ser buscadas.

Nesse contexto, deve-se buscar uma arquitetura simplificada que permita estabelecer uma relação entre as necessidades dos alunos que estão começando a programar e as representações em hardware correspondentes. É necessário realizar a identificação das principais fontes de incompreensão para os estudantes de modo a prover formas de minimizar suas dificuldades. Por exemplo, podem ser citadas algumas relações importantes entre a programação de alto nível e a sua implementação no hardware, sob a forma de conceitos de arquitetura e organização de computadores. Entre essas relações, destacam-se:

- Declaração de variável e alocação de memória;
- Constantes e operandos imediatos;
- Atribuição de variáveis e sua correspondência com as operações de acesso à memória; e
- Operações aritméticas e sua execução no hardware.

Nesse sentido, discussões continuadas entre professores das áreas de Algoritmos e Programação e de Arquitetura de Computadores na Universidade do Vale do Itajaí (UNIVALI) levaram à concepção de uma arquitetura simplificada de processador tendo duas diretrizes principais de projeto:

1. Facilitar o aprendizado da arquitetura e o entendimento dos conceitos por ela ilustrados por alunos da primeira fase do curso de Ciência da Computação da UNIVALI; e
2. Viabilizar e facilitar o uso da arquitetura em disciplinas mais avançadas, promovendo uma integração interdisciplinar.

Para minimizar a complexidade do processador, foi adotada uma arquitetura orientada a acumulador com características derivadas da arquitetura dos microcontroladores PIC® da Microchip (2003). No entanto, algumas escolhas foram feitas no sentido de conferir uma regularidade arquitetural maior do que a desses microcontroladores, nos quais as palavras de dado e de instrução têm tamanhos diferentes e a largura do campo de código de operação pode variar para as diversas classes de instrução. Essas escolhas foram baseadas na assertiva apresentada por Patterson e Hennessy (2005) de que quanto mais regular for a arquitetura de um processador, mais fácil será a sua implementação. Além disso, entende-se que a regularidade também favorece o entendimento da arquitetura do processador.

Disso, foi realizada uma especificação arquitetural de uma família de processadores simplificados denominada BIP (Basic Instruction-set Processor) contendo três modelos de processador, cada um com suporte incremental ao entendimento de conceitos de Algoritmos e Programação e oferecendo recursos adicionais para seu uso em outras disciplinas, em uma abordagem interdisciplinar:

- BIP I: inclui apenas instruções de aritmética e de acesso à memória de dados, tendo como foco o suporte ao entendimento de conceitos como níveis de linguagem, constantes, variáveis, representação de dados e de instruções, conjuntos de instruções, programação em linguagem de montagem e geração de código em linguagem de máquina;
- BIP II: acrescenta instruções de desvio, com foco na inclusão de suporte aos conceitos de estruturas de controle para desvios condicionais e incondicionais e laços de repetição; e
- BIP III: acrescenta instruções de lógica, focando na inclusão de suporte a operações de lógica bit-a-bit.

A seguir, é apresentada a especificação da família BIP, identificando-se seus atributos arquiteturais e explicando o porquê de cada escolha tomada.

6.2.1 Atributos Arquiteturais da Família BIP

Com base nas diretrizes de projeto discutidas previamente, foram definidos os seguintes atributos para a arquitetura da Família BIP:

1. Palavras de instrução e de dados: Os tamanhos das palavras de instrução e de dado foram fixados em um mesmo valor (16 bits) para viabilizar implementações baseadas em uma única memória para o armazenamento de instruções e de dados (arquitetura de von Neumann) e em memórias separadas (arquitetura Harvard);
2. Registradores: A arquitetura prevê quatro registradores: PC (Program Counter), IR (Instruction Register), STATUS e ACC, sendo que o uso de alguns desses registradores varia conforme a implementação do processador. Por exemplo, numa implementação monociclo com arquitetura Harvard o PC aponta para a instrução corrente e o IR não é necessário, pois a saída da memória de instruções mantém a instrução a ser executada durante todo o ciclo da instrução. Já numa implementação multiciclo, o IR deve ser usado para armazenar a instrução corrente, pois o PC é atualizado durante o ciclo de instrução para apontar para a próxima instrução a ser executada. O registrador STATUS inclui *flags* referentes à execução da última instrução: Z (Zero), N (Negative) e C (Carry). Finalmente, o registrador ACC (o acumulador) armazena o resultado da última operação aritmética;
3. Modelos de execução: Pelo fato de ter um único registrador de propósito geral (o ACC), foi considerado que as operacões aritméticas e lógicas poderiam processar operandos oriundos da memória de dados (o que também ocorre nos microcontroladores PIC®). Dessa forma, são suportados os modelos de execução Registrador-Registrador e Registrador-Memória;

4. Formato de instrução: Foi definido um único formato de representação para todo o conjunto de instruções. Esse formato, ilustrado na Figura 6.1, é composto por dois campos: um código de operação de 5 bits e um operando explícito de 11 bits;

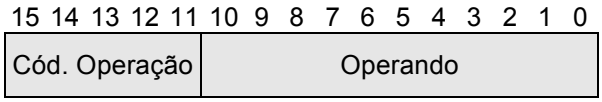


Figura 6.1. Formato de instrução

5. Espaços de endereçamento: O espaço de endereçamento é limitado a 2048 posições devido à largura do campo de operando da instrução (11 bits). A arquitetura utiliza entrada-e-saída (E/S) mapeada em memória, sendo que o acesso à E/S é realizado usando as mesmas instruções de acesso à memória. A organização dos espaços de endereçamento pode ser explorada de diferentes maneiras. Pode ser definido um espaço de endereçamento único para instruções, dados e E/S, conforme ilustrado na Figura 6.2.a. Porém, se o processador utilizar uma organização de memória tipo Harvard, podem ser definidos dois espaços de endereçamento separados: um para instruções e outro para dados e E/S (como é ilustrado na Figura 6.2.b). Essa abordagem permite implementar programas com até 2K instruções e 2K posições de dados e de E/S;

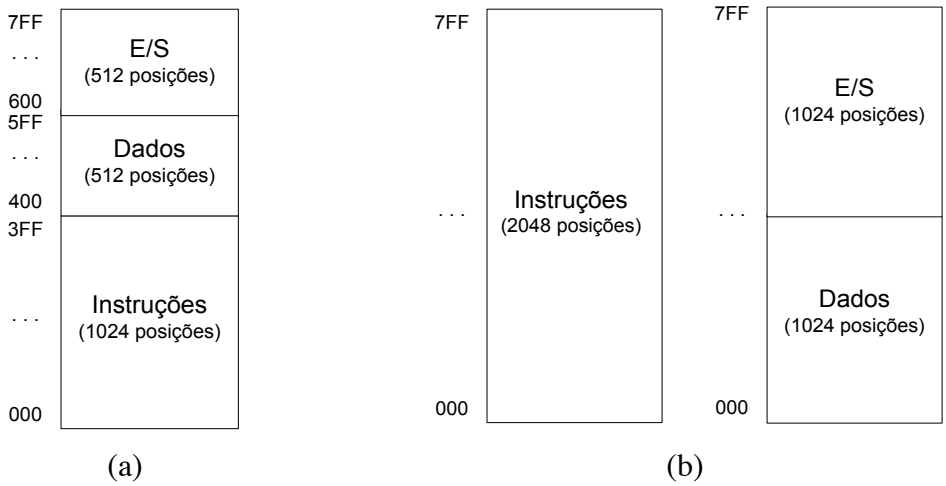


Figura 6.2. Alternativas de organização do espaço de endereçamento: (a) unificado; (b) dividido

6. Modos de endereçamento: Foram definidos dois modos de endereçamento: imediato e direto. O modo imediato (ilustrado na Figura 6.3) é utilizado para operações envolvendo um registrador (o ACC ou o PC) e o operando da instrução (neste caso uma constante de 11 bits com o sinal representado em complento de dois). O modo de endereçamento direto é utilizado para operações entre o acumulador e uma posição do espaço de endereçamento de memória apontada pelo operando da instrução, como ilustrado na Figura 6.4.

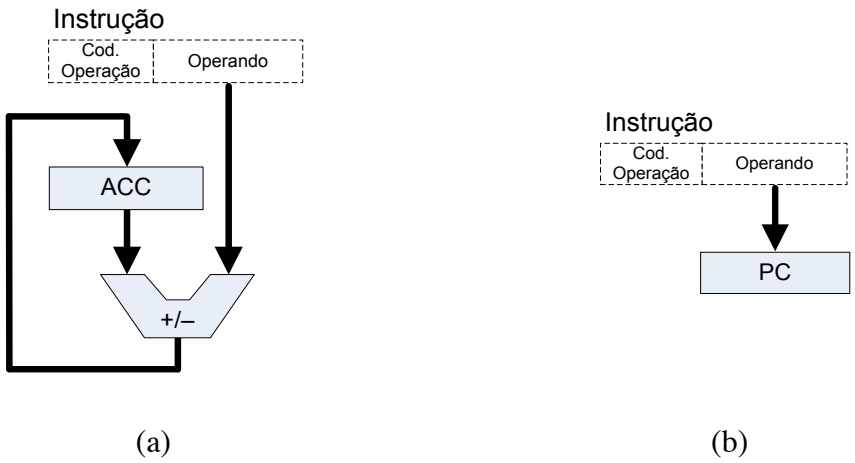


Figura 6.3. Modo de endereçamento imediato: (a) operação com o ACC; (b) operação com o PC

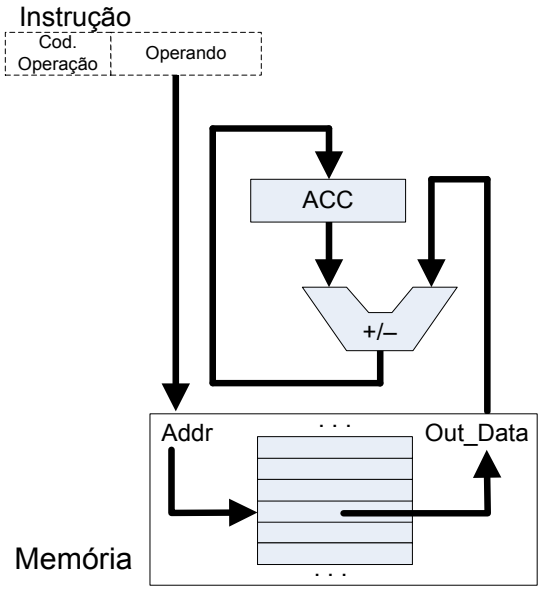


Figura 6.4. Modo de endereçamento direto

6.2.2 Arquitetura e Organização do BIP I

O conjunto de instruções do BIP I é formado por oito instruções, incluindo instruções de controle (HLT), armazenamento em memória (STO), carga no acumulador (LD e LDI) e de aritmética (ADD, ADDI, SUB e SUBI), as quais são descritas na Tabela 6.1, a seguir.

Tabela 6.1. Conjunto de instruções do BIP I

Código da Operação	Instrução		Operação	Classe
00000	HLT		Paralisa a execução do programa	Controle
00001	STO	operando	Memória[operando]← ACC	Armazenamento
00010	LD	operando	ACC ← Memória[operando]	Carga
00011	LDI	operando	ACC ← operando	Carga
00100	ADD	operando	ACC ← ACC + Memória[operando]	Aritmética
00101	ADDI	operando	ACC ← ACC + operando	Aritmética
00110	SUB	operando	ACC ← ACC – Memória[operando]	Aritmética
00111	SUBI	operando	ACC ← ACC – operando	Aritmética

A instrução HLT (*halt*) tem a função de desabilitar a atualização do PC, paralisando a execução do programa. Nas demais instruções, o PC é incrementado em uma unidade.

A instrução STO (*store*) realiza a transferência do conteúdo do registrador ACC para uma posição do espaço de endereçamento de memória (Memória[operando]). Quanto às instruções de carga, a instrução LD (*load*) realiza uma operação de transferência de uma posição do espaço de endereçamento de memória para o acumulador, enquanto que a instrução LDI (*load immediate*) carrega uma constante (o operando) no acumulador.

Com relação às instruções aritméticas, são disponibilizadas instruções de soma e de subtração entre o acumulador e uma posição do espaço de endereçamento de memória e entre o acumulador e uma constante. São elas: ADD (*add*), SUB (*subtract*), ADDI (*add immediate*) e SUBI (*subtract immediate*). Embora a instrução SUBI possa ser dispensada pelo uso do ADDI com um operando negativo, como é feito no MIPS [Patterson e Hennessy 2005], optou-se por disponibilizar a instrução SUBI para conferir uma maior facilidade ao aprendizado.

Quanto aos modos de endereçamento, as instruções LDI, ADDI e SUBI utilizam o modo imediato, enquanto que as instruções STO, LD, ADD e SUB utilizam o modo direto. Em todas essas instruções, o acumulador é utilizado como um operando implícito, atuando como operando fonte e/ou destino das operações realizadas.

Na Tabela 6.2, são apresentados alguns exemplos de uso da linguagem de montagem do BIP I para a implementação de abstrações representadas em linguagens de alto nível. Como pode ser observado, pelo seu conjunto de instruções, o BIP I consiste basicamente de uma calculadora programável que realiza operações de soma e subtração com variáveis e constantes. No entanto, apesar de limitado, esse conjunto de instruções permite ilustrar várias relações entre as abstrações estudadas nas disciplinas da área de Algoritmos e Programação e sua representação no nível arquitetural do processador, conforme será discutido posteriormente.

Tabela 6.2. Uso da linguagem de montagem do BIP I

Abstração	Código de alto nível	Código na linguagem de montagem
Atribuição de uma constante	`A = 10;`	`LDI 10 ;ACC ← 10` `STO A ;A ← ACC`
Atribuição de uma variável	`A = B;`	`LD B ;ACC ← B` `STO A ;A ← ACC`

Comando com uma operação aritmética	`A = A + 1;`	`LD A ;ACC ← A`
		`ADDI 1 ;ACC ← ACC + 1`
		`STO A ;A ← ACC`

Comando com múltiplas operações aritméticas	`A = A + B - 3;`	`LD A ;ACC ← A`
		`ADD B ;ACC ← ACC + B`
		`SUBI 3 ;ACC ← ACC - 3`
		`STO A ;A ← ACC`

Na Figura 6.5, são apresentadas duas alternativas de organização para o BIP I. A primeira é uma organização monociclo baseada em uma arquitetura Harvard, com memórias separadas para armazenar instruções e dados (Figura 6.5.a). Já a segunda é uma organização multiciclo baseada em uma arquitetura de von Neumann, com uma memória unificada (Figura 6.5.b). Os diagramas apresentados abstraem alguns aspectos de implementação, como o uso de multiplexadores, a largura dos canais e os sinais de controle (os quais serão detalhados posteriormente).

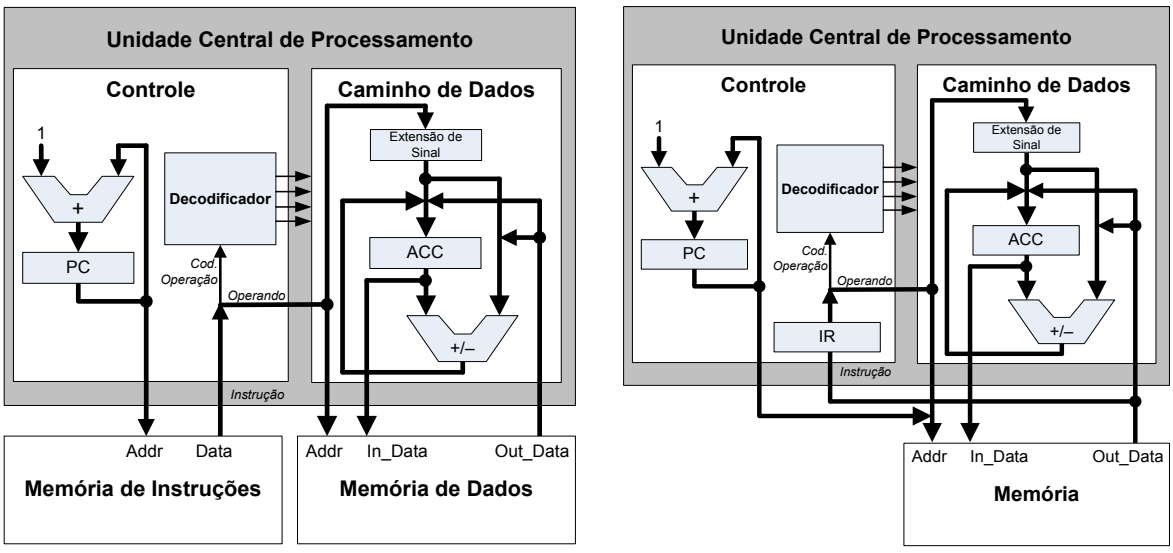


Figura 6.5. Organizações alternativas: (a) Harvard monociclo; (b) von Neumann multiciclo

A Unidade Central de Processamento (UCP) do BIP I é estruturada em dois blocos: o Controle e o Caminho de Dados. O Controle inclui o registrador PC, um somador para atualizar o valor do PC e o decodificador que gera os sinais de controle necessários para a execução de cada instrução. O Caminho de Dados, por sua vez, inclui o registrador ACC, uma unidade aritmética de soma e de subtração e um circuito para estender o sinal

do operando (de 11 para 16 bits). A principal diferença entre a UCP das duas organizações está no uso do registrador IR no Controle. A organização de Harvard dispensa esse registrador pois a memória de instrução mantém em sua saída a instrução corrente. Já a organização de von Neumann precisa do IR pois, após a busca da instrução, a memória pode ser utilizada com fonte ou destino de alguma operação. As duas organizações não incluem o registrador STATUS no Caminho de Dados, pois o conjunto de instruções do BIP I não possui instruções que analisem esse registrador.

Além das organizações ilustradas, outras variações podem ser implementadas. Por exemplo, adicionando o registrador IR à organização Harvard e com as devidas modificações no bloco de decodificação, pode ser construída uma organização com um *pipeline* de dois estágios sobrepondo a busca e a execução de duas instruções subsequentes, assim como é feito nos microcontroladores PIC® da Microchip (2003).

A Figura 6.6 apresenta um detalhamento da organização Harvard monociclo. Nela, podem ser observados os multiplexadores de seleção do Caminho de Dados, os sinais de controle gerados pelo Decodificador e a largura dos canais que interligam os componentes. Os circuitos utilizados na implementação da UCP são aqueles tipicamente estudados em disciplinas de Eletrônica Digital (decodificador, registrador, somador, somador/subtrator, multiplexador e memória). A exceção é bloco de extensão de sinal que converte o canal de 11 bits do operando para a largura da palavra do Caminho de Dados (16 bits). Essa conversão é feita pela atribuição do bit sinal do operando (bit 10) aos bits 11-15 da interface de saída do bloco.

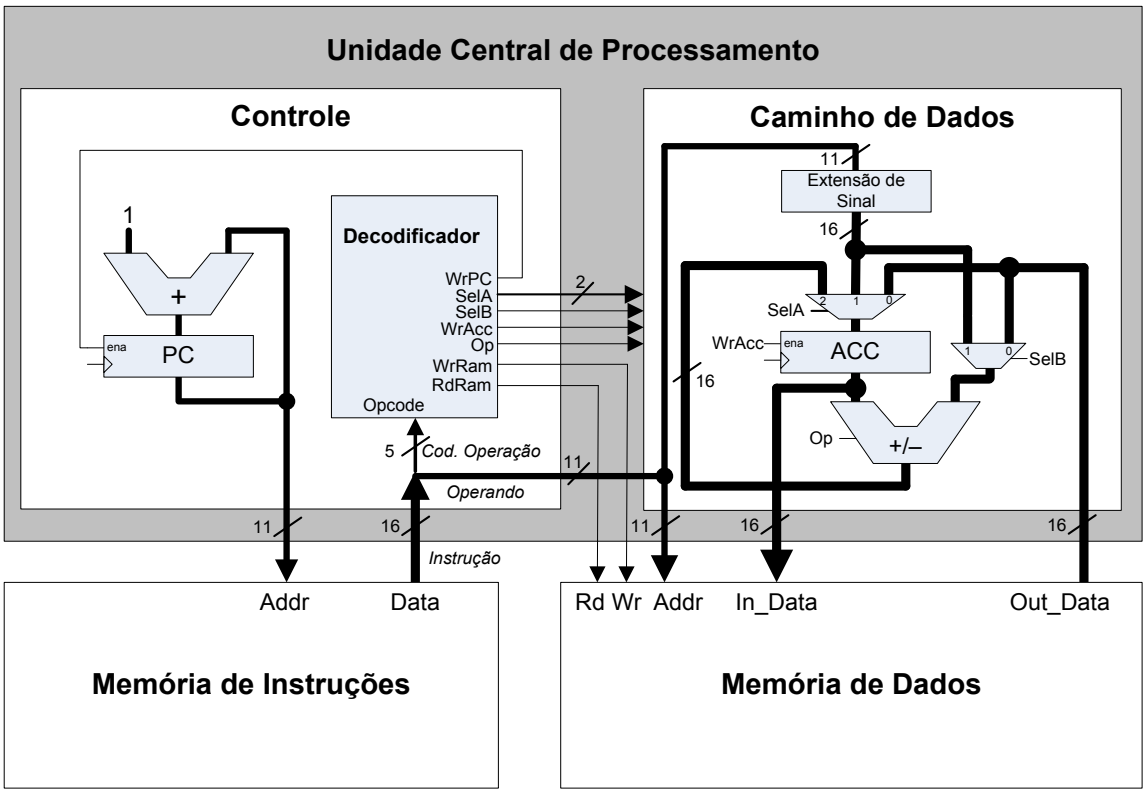


Figura 6.6. Organização monociclo do BIP I baseada em uma arquitetura tipo Harvard

Vale destacar que as organizações ilustradas não incluem uma interface de E/S. No entanto, por utilizar entrada-e-saída mapeada em memória, periféricos e dispositivos de

E/S podem ser facilmente acrescentados conectando-os ao barramento da memória de dados.

6.2.3 Arquitetura e Organização do BIP II

O BIP II é uma extensão do BIP I e possui as mesmas características arquiteturais. Ele foi especificado para suportar estruturas de controle usadas em desvios e em laços de repetição. Para tal, o conjunto de instruções do BIP I foi estendido incluindo seis instruções de desvio condicional: BEQ (*branch on equal*), BNE (*branch on not equal*), BGT (*branch on greater than*), BGE (*branch on greater than or equal*), BLT (*branch on less than*) e BLE (*branch on less than or equal*). Também foi adicionada a instrução de desvio incondicional JMP (*jump*). Essas novas instruções são descritas na Tabela 6.3.

Tabela 6.3. Conjunto de instruções estendido do BIP II

Código da Operação	Instrução	Operação	Classe
01000	BEQ operando	Se (STATUS.Z=1) então PC ← operando Se não PC ← PC + 1	Desvio condicional
01001	BNE operando	Se (STATUS.Z=0) então PC ← operando Se não PC ← PC + 1	Desvio condicional
01010	BGT operando	Se (STATUS.Z=0) e (STATUS.N=0) então PC ← operando Se não PC ← PC + 1	Desvio condicional
01011	BGE operando	Se (STATUS.N=0) então PC ← operando Se não PC ← PC + 1	Desvio condicional
01100	BLT operando	Se (STATUS.N=1) então PC ← operando Se não PC ← PC + 1	Desvio condicional
01101	BLE operando	Se (STATUS.Z=1) ou (STATUS.N=1) então PC ← operando Se não PC ← PC + 1	Desvio condicional
01110	JMP operando	PC ← operando	Desvio incondicional

Conforme visto na Tabela 6.3, para suportar as instruções de comparação e desvio condicional, foi necessário incluir o registrador STATUS, em especial os *flags* booleanos Z (*zero*) e N (*negative*). Qualquer instrução de comparação e desvio condicional deve ser precedida por uma instrução de subtração (SUB ou SUBI). Dependendo do resultado dessa operação, os *flags* Z (Zero) e N (Negative) do registrador STATUS são definidos em 0 (FALSE) ou em 1 (TRUE). As instruções de desvio condicional então verificam o valor desses *flags* para determinar se o desvio deve ser tomado ou não, conforme o tipo de comparação associado. Essa estratégia se deve à limitação do formato de instrução, o qual suporta um único operando explícito. Esse operando é usado para codificar o endereço de desvio, enquanto que os valores comparados são processados pela instrução de subtração.

As instruções de desvio utilizam o modo de endereçamento imediato quando carregam o endereço de desvio no PC, o qual é considerado um operando implícito.

As tabelas a seguir apresentam exemplos de uso do conjunto de instruções do BIP II para implementar as estruturas de controle. Nas tabelas, utiliza-se o termo "*Bloco i*" para designar um segmento de código que representa uma sequência de instruções que executa alguma tarefa, a qual é irrelevante para o contexto do exemplo.

Tabela 6.4. Uso da linguagem de montagem do BIP II (Parte I)

Abstração	**Código de alto nível**	**Código na linguagem de montagem**

Teste de condição do tipo `if-then`	`if (A==B) {`
	`  // Bloco 1`
	`}`
	`// Bloco 2`
		`  LD  A  ;ACC ← A`
		`  SUB  B  ;ACC ← ACC - B`
		`  BNE  L1`
		`  ;Bloco 1`
		`L1:`
		`  ;Bloco 2`

Teste de condição do tipo `if-then-else`	`if (A==B) {`
	`  // Bloco 1`
	`} else {`
	`  // Bloco 2`
	`}`
	`// Bloco 3`
		`  LD  A  ;ACC ← A`
		`  SUB  B  ;ACC ← ACC - B`
		`  BNE  L1`
		`  ;Bloco 1`
		`  JMP  L2`
		`L1:`
		`  ;Bloco 2`
		`L2:`
		`  ;Bloco 3`

Laço de repetição do tipo `while`	`i = 0;`
	`while (i<10) {`
	`  // Bloco 1`
	`  i++;`
	`}`
	`// Bloco 2`
		`  LDI  0  ;ACC ← 0`
		`  STO  I  ;I  ← ACC`
		`L1:`
		`  SUBI 10  ;ACC ← ACC - 10`
		`  BGE  L2`
		`  ;Bloco 1`
		`  LD  I  ;ACC ← I`
		`  ADDI 1  ;ACC ← ACC + 1`
		`  STO  I  ;I  ← ACC`
		`  JMP  L1`
		`L2:`
		`  ;Bloco 2`

Tabela 6.5. Uso da linguagem de montagem do BIP II (Parte II)

Abstração	Código de alto nível	Código na linguagem de montagem

Laço de repetição do tipo `do-while`	`i = 0;`
	`do {`
	`  // Bloco 1`
	`  i++;`
	`} while (i<10)`
	`// Bloco 2`
		`  LDI  0  ;ACC ← 0`
		`  STO  I  ;I  ← ACC`
		`L1:`
		`  ;Bloco 1`
		`  LD  I  ;ACC ← I`
		`  ADDI 1  ;ACC ← ACC + 1`
		`  STO  I  ;I  ← ACC`
		`  SUBI 10  ;ACC ← ACC - 10`
		`  BLT  L1`
		`  ;Bloco 2`

Laço de repetição do tipo `for`	`for(i=0;i<10;i++){`
	`  // Bloco 1`
	`}`
	`// Bloco 2`
		Implementação em linguagem de montagem idêntica à implementação do laço de repetição do tipo `while`

A Figura 6.7 apresenta a organização Harvard monociclo do processador BIP II. Em relação à organização do processador BIP I, ilustrada previamente na Figura 6.6, essa organização inclui os circuitos necessários à implementação das instruções de desvio: (*i*) um multiplexador na entrada do PC, para permitir a carga do valor do operando; e (*ii*) o registrador STATUS, com os *flags* N e Z.

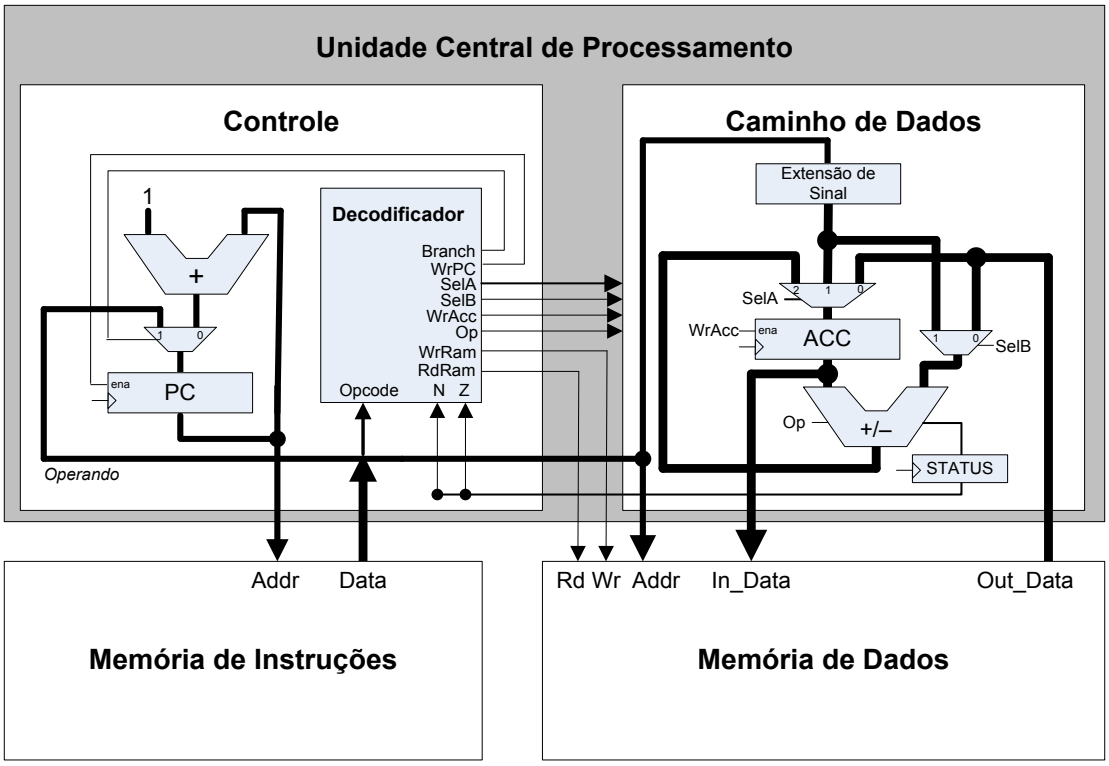


Figura 6.7. Organização monociclo do BIP II baseada em uma arquitetura tipo Harvard

6.2.4 Arquitetura e Organização do BIP III

O BIP III estende o conjunto de instruções do BIP II adicionando nove instruções de lógica bit-a-bit, as quais são descritas na Tabela 6.6.

Tabela 6.6. Conjunto de instruções estendido do BIP III

Código da Operação	Instrução	Operação	Classe
01111	NOT	ACC ← NOT(ACC)	Lógica
10000	AND operando	ACC ← ACC AND Memória[operando]	Lógica
10001	ANDI operando	ACC ← ACC AND operando	Lógica
10010	OR operando	ACC ← ACC OR Memória[operando]	Lógica
10011	ORI operando	ACC ← ACC OR operando	Lógica
10100	XOR operando	ACC ← ACC XOR Memória[operando]	Lógica
10101	XORI operando	ACC ← ACC XOR operando	Lógica
10110	SLL operando	ACC ← ACC << operando	Lógica
10111	SRL operando	ACC ← ACC >> operando	Lógica

A organização do BIP III é similar à organização do BIP II. As poucas modificações necessárias referem-se à substituição do somador-subtrator do caminho de dados por uma unidade funcional integrando uma UAL (Unidade Aritmética Lógica) e *barrel shifters* para realizar os deslocamentos de bits à direita e à esquerda. Além disso, o Decodificador deve ser atualizado para identificar as novas instruções. A Tabela 6.7, logo a seguir, apresenta a codificação definida para o sinal *Op* que comanda a Unidade Funcional, o qual passa a ter 3 bits de largura no BIP III.

Tabela 6.7. Operações suportadas pela unidade funcional do BIP III

Op	Operação	Instruções
000	Soma	ADD e ADDI
001	Subtração	SUB e SUBI
010	Função lógica E	AND e ANDI
011	Função lógica OU	OR e ORI
100	Função lógica XOR	XOR e XORI
101	Função lógica NOT	NOT
110	Deslocamento lógico para a esquerda	SLL
111	Deslocamento lógico para a direita	SRL

6.3 Utilização do BIP no Ensino

As iniciativas de criação de processadores com objetivo didático normalmente buscam apoiar as disciplinas diretamente envolvidas, como Arquitetura e Organização de Computadores. A criação da família de processadores BIP buscou transcender esse enfoque, induzindo um leque de possibilidades de exploração dos conceitos envolvidos em diferentes disciplinas da Ciência da Computação em uma abordagem interdisciplinar.

As três versões do processador BIP permitiram trabalhar níveis de complexidade diferentes e com isso envolver disciplinas do primeiro, segundo, terceiro e sétimo semestres do curso de Ciência da Computação da Universidade do Vale do Itajaí (UNIVALI), além de envolver trabalhos de conclusão de curso.

Sabe-se que existem variações na matriz curricular e na nomenclatura de disciplinas nos cursos de Ciência da Computação de instituições distintas. Tendo isso em vista, a Tabela 6.8 apresenta um extrato da organização curricular utilizada no curso de Ciência da Computação da UNIVALI. Foram acrescidas informações sobre as ementas das disciplinas em que os conceitos do BIP foram trabalhos a fim de permitir ao leitor estabelecer uma relação destas com as disciplinas correlatas em outras instituições que possuam o curso de Ciência da Computação.

Tabela 6.8. Extrato da matriz curricular do curso de Ciência da Computação da Universidade do Vale do Itajaí – UNIVALI

1º Período		
Disciplina	**C/H**	**Ementa**
Algoritmos e Programação	120	Conceitos preliminares. Manipulação de dados. Estruturas de controle de fluxo. Tipos compostos de dados. Modularização de algoritmos. Programação em linguagem C.
Computação Básica	60	Histórico da computação. Sistemas de numeração. O hardware do computador. O software do computador. Funcionamento básico do computador.
2º Período		
Disciplina	**C/H**	**Ementa**
Circuitos Digitais	60	Circuitos com portas lógicas. Simplificação de circuitos. Circuitos combinacionais. Circuitos sequenciais.
3º Período		
Disciplina	**C/H**	**Ementa**
Arquitetura e Organização de Computadores I	60	Introdução à arquitetura e organização de computadores. Conjunto de instruções e programação em linguagem de montagem. Representação de dados e aritmética binária. Organização de processadores. Processamento em Pipeline.
7º Período		
Disciplina	**C/H**	**Ementa**
Compiladores	60	Visão geral de um compilador. Análise léxica. Análise sintática. Análise semântica. Geração de código. Tópicos complementares em compiladores.

A seguir são relatadas as experiências dos autores na utilização dos conceitos relacionados ao uso do BIP nas referidas disciplinas.

6.3.1 Algoritmos e Programação

A crença inicial que motivou o enfoque interdisciplinar na concepção da família de processadores BIP foi de que utilizar um modelo simplificado de computador auxilia a melhorar a compreensão de conceitos relacionados à aprendizagem inicial de programação. Entende-se que, ao reduzir o grau de abstração envolvido com a aprendizagem de conceito ligados ao uso de memória, operações aritméticas, desvios e laços, é possível facilitar a aprendizagem dos alunos que em geral apresentam problemas em lidar com abstrações.

As relações entre as abstrações estudadas nas disciplinas da área de Programação e sua representação no nível arquitetural do processador são listadas na Tabela 6.9.

Tabela 6.9. Relações entre conceitos de Programação e de Arquitetura suportadas pelo BIP I

Conceitos de Programação	Conceitos de Arquitetura de Computadores
Variável	Posição na memória
Constante	Operando imediato na instrução
Atribuição	Acesso à memória para leitura e/ou escrita de/em uma posição
Operação aritmética	Utilização de unidade de soma/subtração
Comandos com múltiplas operações	Uso de uma instrução para cada operação realizada
Desempenho dos programas	Número de instruções na linguagem de montagem
Papel do compilador	Tradução da linguagem de alto nível para a linguagem de montagem

A abordagem utilizada anteriormente à existência do BIP resumia-se a uma breve explicação sobre o funcionamento de um computador, ilustrando os componentes básicos da arquitetura de Von Neumann e em seguida realizava-se a introdução de conceitos como váriáveis, atribuições e operações de entrada-e-saída.

Com a introdução do BIP I, a explicação dos conceitos básicos de arquitetura de computadores tornou-se mais detalhada e os conceitos apresentados não restringiram-se apenas à disciplina de Algoritmos e Programação. Foram mostrados exemplos de programas em português estruturado (Portugol) e os correspondentes na linguagem de montagem. O Portugol vem sendo adotado nesta disciplina como uma linguagem de programação simplificada visando reduzir a complexidade na criação dos primeiros programas eliminando a dificuldade que alguns alunos apresentam com o idioma inglês e com detalhes específicos da interface dos compiladores e ambientes de programação comerciais [Hostins e Raabe, 2007].

Posteriormente, foram realizados exercícios com os alunos sobre a construção de programas em linguagem de montagem. A seguir, são apresentados exemplos de exercícios com as respostas esperadas em destaque:

Exemplo de Exercício 1

Dado o trecho de código a seguir, escrito na linguagem de montagem do BIP, comente cada linha do código identificando a operação realizada:

```
LDI   0     ; ACC ← 0
ADDI  1     ; ACC ← ACC + 1
ADD   B     ; ACC ← ACC + B
STO   A     ; A   ← ACC
```

Resposta esperada

Exemplo de Exercício 2

Dado o trecho de código a seguir, escrito em linguagem de alto nível, escreva o código equivalente na linguagem de montagem do BIP, identificando, nos comentários, a operação realizada por cada comando na linguagem de montagem:

Código em linguagem de alto nível

```
X = 0;
Y = 2;
X = X + Y;
```

Código na linguagem de montagem do BIP:

```
LDI   0     ; ACC ← 0
STO   X     ; X   ← ACC
LDI   2     ; ACC ← 02
STO   Y     ; Y   ← ACC
ADD   X     ; ACC ← ACC + X
STO   X     ; X   ← ACC
```

Resposta esperada

Exemplo de Exercício 3

Dado o trecho de código a seguir, escrito na linguagem de montagem do BIP, comente cada linha de código e obtenha o código equivalente em linguagem de alto nível:

Código na linguagem de montagem do BIP:

```
LD    B     ; ACC ← B
ADDI  1     ; ACC ← ACC + 1
SUB   C     ; ACC ← ACC - C
STO   A     ; A   ← ACC
```

Código em linguagem de alto nível:

```
A = B + 1 - C;
```

Resposta esperada

Após a apresentação da arquitetura do BIP I e a aplicação de exercícios similares aos exemplificados acima, os principais aspectos observados com relação à aprendizagem dos alunos foram:

- Os alunos demonstraram alguma dificuldade inicial em compreender as instruções de linguagem de montagem, mas rapidamente passaram a adquirir fluência na resolução dos exercícios;
- A compreensão da existência do acumulador tornou a operação de atribuição mais clara. O que acontecia com o resultado intermediário da operação A+1 em A ← A + 1 antes ficava obscuro;
- Um problema recorrente em vários semestres, em que alguns alunos declaravam contantes inteiras como se fossem variáveis (Ex: inteiro 5) ou ainda realizavam operações de atribuição para constantes (5 ← 3 + 2), não ocorreu nos semestres em que foi usada a abodagem com o BIP I;
- Como a explicação sobre o BIP I reuniu turmas de períodos diferentes (Algoritmos e Programação do 1º período e Arquitetura e Organização de Computadores I do 3º período) ficou claro aos alunos do primeiro período que os conceitos seriam importantes também para a sequência do curso.

A aplicação do processador BIP I permitiu confirmar a efetividade da abordagem proposta, uma vez que os alunos puderam consolidar os conceitos estudados na disciplina da área de Programação.

No entanto, pôde-se evidenciar que o conjunto de instruções limitado do BIP I restringiu o seu uso por não suportar a implementação de estruturas de controle (desvios condicional e incondicional) no nível arquitetural. Essa limitação, prevista no início do projeto, já foi contornada com a disponibilização da arquitetura do BIP II. Porém esse processador ainda não foi utilizado nesta disciplina. Nas próximas turmas, pretende-se aplicar o BIP I inicialmente para apresentar os conceitos introdutórios e, após o seu entendimento, utilizar o BIP II para ampliar a abrangência dos conceitos abordados.

6.3.2 Computação Básica

Esta disciplina tem por objetivo apresentar uma visão geral da Ciência da Computação aos alunos ingressantes no curso, com ênfase no estudo dos princípios de funcionamento dos sistemas computacionais.

Nesse contexto, o BIP I passou a ser utilizado como processador de referência para ilustrar os conceitos de arquitetura e de organização de computadores, permitindo um melhor entendimento por parte dos alunos a respeito dos atributos arquiteturais de um processador e da diferença entre os conceitos de arquitetura e organização.

No estudo da arquitetura, foram apresentados os principais conceitos relacionados, utilizando o BIP I para exemplificar os diferentes atributos arquiteturais, como, por exemplo, o tamanho da palavra de dados, o tamanho da palavra de instrução, tipos de dados, formato de instrução, modos de endereçamento, registradores e conjunto de instrução. Uma vez feita a explanação desses conceitos e o estudo da arquitetura do BIP I, foram apresentadas as relações entre comandos de alto nível e comandos na linguagem de montagem do BIP I. Após, os alunos realizaram exercícios de fixação similares aos ilustrados na Sub-seção 6.3.1.

Na sequencia, para reduzir o nível de abstração, foram mostrados os códigos binário equivalentes de alguns programas-exemplo, o que permitiu ilustrar o conceito de

linguagem de máquina, processo de geração de código executável e o papel do compilador e do montador nesse processo.

Com relação ao estudo de organização de computadores, o BIP I foi utilizado para ilustrar o funcionamento de um processador. O grau de abstração adotado nesse estudo foi definido considerando que os alunos do primeiro período não possuem conhecimentos sobre circuitos digitais. Inicialmente, foram explicadas as funcionalidades dos blocos construtivos do BIP I, sem preocupação com a sua estrutura interna. Em seguida, foi mostrada a construção do processador para a execução de cada instrução e, por fim, foi apresentado um modelo simplificado da organização do BIP I (ilustrado na Figura 6.8). Foram realizadas atividades de fixação em que os alunos foram orientados a destacar os componentes e os canais do processador utilizados na execução de cada instrução, conforme ilustrado na Figura 6.9.

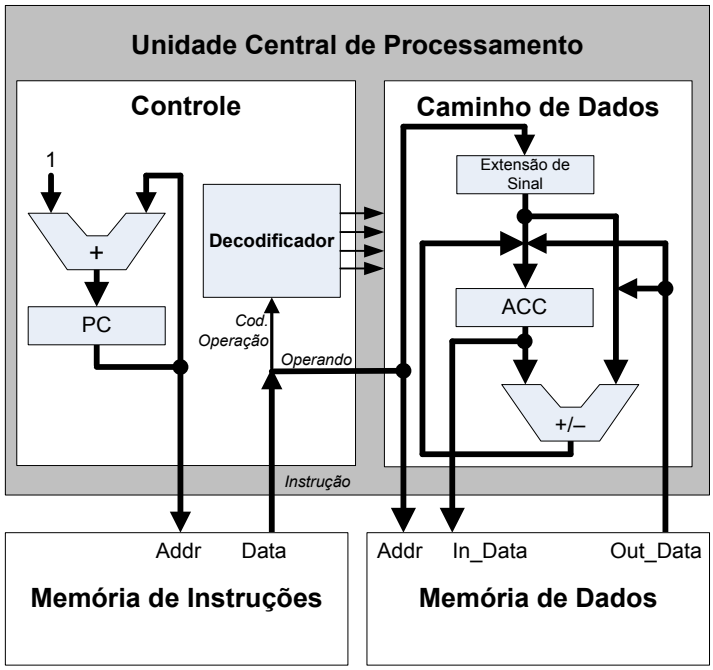


Figura 6.8. Organização Harvard monociclo do BIP I (visão simplificada)

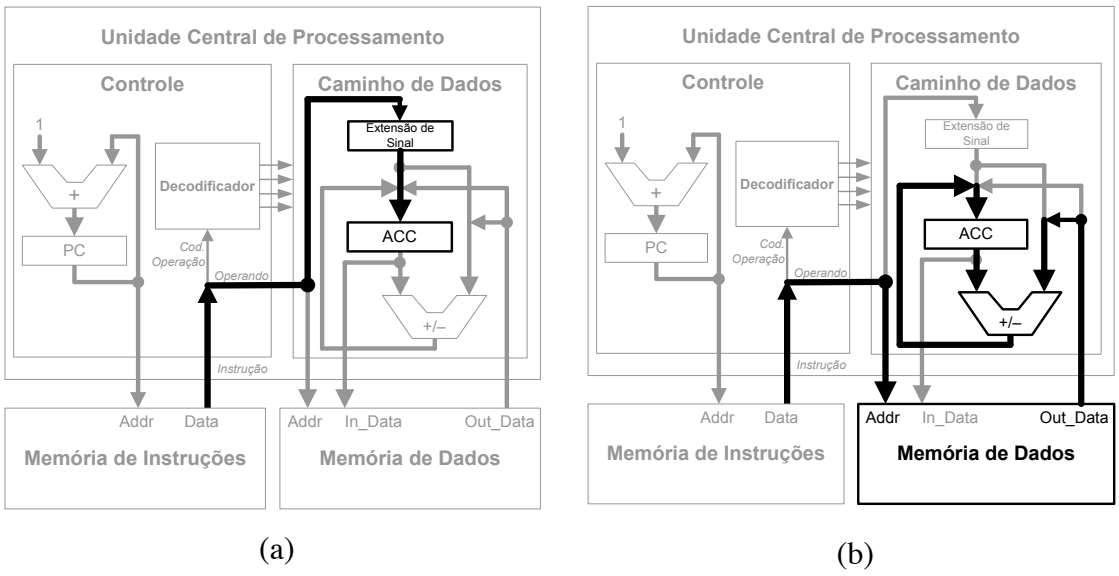


Figura 6.9. Exercício sobre a organização do BIP I: (a) execução da instrução LDI; (b) execução da instrução ADD

A partir dos exercícios de análise da execução das instruções na organização do BIP I, os alunos demonstraram entendimento básico a respeito da operação do processador e dos diferentes níveis de abstração: arquitetural e organizacional.

6.3.3 Circuitos Digitais

Esta disciplina tem por objetivo caracterizar e aplicar os fundamentos da Eletrônica Digital para aplicações em análise de circuitos utilizados em um computador. Nela, o aluno aprende as técnicas necessárias para implementar os blocos construtivos básicos de um processador, como, por exemplo, registradores, somadores, decodificadores e multiplexadores, entre outros.

Nesta disciplina, o BIP I foi utilizado para exemplificar o uso desses blocos construtivos para a construção de um sistema digital do tipo processador programável. Como a disciplina é baseada em ferramentas de captura de esquemático e de simulação lógica, foi aplicada uma atividade na qual os alunos deviam implementar o BIP I utilizando essas ferramentas e analisar o funcionamento do processador através de diagramas de forma de onda produzidos pelo simulador.

Diversos alunos já haviam estudado o BIP I nas disciplinas de Computação Básica e/ou de Algoritmos e Programação, o que facilitou a realização da atividade. No entanto, mesmo para aqueles que não haviam tido a oportunidade de estudar o BIP I, a compreensão acerca da sua arquitetura e da sua organização também foi facilitada pela simplicidade do processador.

O processador permitiu que os alunos vislumbrassem a aplicação dos conceitos estudados na disciplina aproximando-a da disciplina seguinte no curso (Arquitetura e Organização de Computadores). Ao mesmo tempo, essa atividade teve o efeito de um projeto integrador, incentivando os alunos a resgatar conceitos já estudados em outras disciplinas.

6.3.4 Arquitetura e Organização de Computadores I

Esta disciplina tem por objetivo geral ampliar a visão sobre arquitetura e organização de computadores, com vistas à programação na linguagem de montagem de um processador, identificando aspectos do projeto da sua organização. A disciplina utiliza o livro "Organização e Projeto de Computadores" de Patterson e Hennessy (2005) como livro-texto e, portanto, adota o processador MIPS como arquitetura de referência ao longo do semestre.

No entanto, no momento em que os conceitos de arquitetura e de organização começam a ser tratados, os processadores BIP são utilizados como referência inicial para ilustrar esses conceitos e resgatar o aprendizado obtido em disciplinas anteriores do Curso. Posteriormente, ao se realizar o estudo do MIPS, é feita uma análise comparativa das características arquiteturais e organizacionais dos dois processadores. Procura-se também discutir as diretrizes de projeto do BIP, demonstrando que o foco na simplicidade do processador lhe impõe limitações quando comparado a um processador cujo projeto é focado no desempenho, como o MIPS.

O papel do BIP nesta disciplina é diferente do seu papel nas disciplinas das fases iniciais. A sua contribuição está em permitir um aumento incremental da complexidade dos temas abordados ao longo do semestre. Como a arquitetura do MIPS não foi especificada visando o aprendizado, mas sim a facilidade de implementação do processador e o seu desempenho, iniciar o estudo dos conceitos de arquitetura e de organização a partir do BIP tende a facilitar o entendimento inicial desses conceitos.

6.3.5 Compiladores

A utilização da Família BIP na disciplina de compiladores proporcionou uma alternativa para demonstrar todas as fases de construção do compilador seguindo a abordagem sugerida por Lins (2000), na qual inicia-se definindo a arquitetura da linguagem alvo para então se definir os aspectos sintáticos e léxicos da linguagem de alto nível.

O conjunto de instruções do BIP mostrou-se bastante simples, permitindo a geração direta do código através de ações semânticas sem passar por linguagens intermediárias, conforme ilustra a Tabela 6.10. Essa característica possibilitou a substituição da linguagem intermediária que vinha sendo utilizada até então (Linguagem da Máquina Virtual de Pilha), viabilizando à maioria dos alunos concluir a etapa de geração de código (o que raramente ocorria em semestres anteriores).

Para ilustrar melhor a simplicidade do processo de geração de código, na Tabela 6.10, é demonstrada a geração de um desvio condicional para a linguagem Portugol. Foi utilizada a ferramenta ANTLR3 (Parr, 2009) para apoiar a geração do analisador léxico e sintático.

Tabela 6.10. Ações Semânticas para a geração de desvios condicionais

Portugol	**Ações Semânticas**	***Assembly***

`Se (x > 8) entao`
`  x <- 0`
`fimse`
	1. cmd_se: SE expRelacional
	2. {
	3. add_instrucao("LD", resultado_Esq);
	4. add_instrucao("SUB",resultado_Dir);
	5. add_instrucao(opRelacional,"ENDIF1");
	6. rotulos.Push("ENDIF1");
	7. }
	8. ENTAO lista_cmdo FIMSE
	9. {
	10. add_instrucao(rotulos.Pop());
	11. }
	12. ;
		`  LD  x`
		`  STO $100`
		`  LDI 8`
		`  STO $101`
		`  LD  $100`
		`  SUB $101`
		`  BLE ENDIF1`
		`  LDI 0`
		`  STO x`
		`ENDIF1:`

Para a realização do desvio condicional, inicialmente, são resolvidas as expressões dos lados esquerdo e direito do operador relacional. Os resultados dessas expressões são armazenados em variáveis temporárias e, posteriormente, o resultado da expressão do lado direito é subtraído do valor resultante da expressão do lado esquerdo do operador (linhas 3 e 4). As instruções de desvio do BIP baseiam-se na execução prévia de uma instrução de subtração entre os valores comparados e na análise do registrador STATUS que armazena informações sobre o resultado da última operação. De acordo com o valor resultante dessa subtração, a instrução de desvio condicional determina se o desvio deve ser tomado ou não.

A instrução de desvio é gerada de acordo com o operador relacional utilizado (linha 4). O nome do rótulo é colocado em uma pilha (linha 5) para utilização ao final da expressão (linha 9).

O método `add_instrucao` (linha 10) adiciona a instrução em uma estrutura de dados que será utilizada para a geração de código e posterior uso na simulação do programa.

Alguns trabalhos iniciados na disciplina tiveram continuidade na forma de trabalhos de conclusão de curso. Uma das ferramentas desenvolvidas (descrita a seguir) consistiu de um simulador que será utilizado futuramente na disciplina.

6.3.6 Trabalhos de Conclusão de Curso

Dois trabalhos de conclusão de curso relacionados ao BIP já foram realizados no Curso de Ciência da Computação da UNIVALI. Em um deles, foi desenvolvido um microcontrolador baseado na arquitetura BIP (o μBIP). Em outro, foi desenvolvida uma IDE (Integrated Development Environment) denominada BipIde que inclui simuladores do BIP I e do BIP II. Esses trabalhos são relatados resumidamente a seguir.

6.3.6.1 μBIP

Visando estender o uso do BIP para disciplinas avançadas nas áreas de projeto de sistemas digitais e de sistemas embarcados, foi desenvolvido um microcontrolador baseado na arquitetura BIP, o qual foi denominado *μ*BIP (lê-se *microbip*)

[Pereira 2008][Pereira e Zeferino 2008]. Esse microcontrolador estende as características dos processadores BIP, agregando novas instruções, periféricos e funcionalidades típicas de microcontroladores.

Para a especificação deste microcontrolador, foi feita uma pesquisa junto a professores e engenheiros de desenvolvimento de hardware de empresas de base tecnológica. Essa consulta permitiu definir as características de uma primeira versão do microcontrolador e estabelecer um *roadmap* para futuras versões.

Para essa primeira versão, foram selecionadas e implementadas as seguintes características:

- Suporte a endereçamento de vetores;
- Suporte a chamadas de procedimentos;
- Suporte a interrupções;
- Portas de E/S bidirecionais; e
- Temporizador programável com *prescaler*.

Para tal, foi necessário incluir hardware adicional no processador, como um registrador para endereçamento indireto de vetores (INDR), uma pilha em hardware para salvamento de contexto e um controlador de interrupções, além dos periféricos selecionados. A pilha em hardware é usada para salvar o contexto do programa nas chamadas de procedimentos e no atendimento de uma interrupção. Utilizando-se uma abordagem similar à adotada nos microcontroladores PIC® da Microchip (2003), apenas o valor de PC+1 é salvo na pilha, cujo topo (ToS – Top-of-Stack) é apontado pelo registrador SP (Stack Pointer).

A Figura 6.10 ilustra a organização do μBIP. Nessa figura, podem ser observados alguns dos novos componentes de hardware, sendo que os periféricos estão representados pelo bloco SFR (Special Function Registers).

O conjunto de instruções do BIP III foi estendido para oferecer o suporte arquitetural para uso desses recursos. As novas instruções, descritas na Tabela 6.11, incluem: (*i*) duas instruções para manipulação de vetor baseadas no modo de endereçamento indireto – STOV (*store vector*) e LDV (*load vector*); (*ii*) uma instrução de chamada de procedimentos (CALL); e (*iii*) duas instruções de retorno de procedimento (RETURN e RETINT), sendo que RETINT (*return from interrupt*) serve para realizar o retorno de rotinas de atendimento a interrupções.

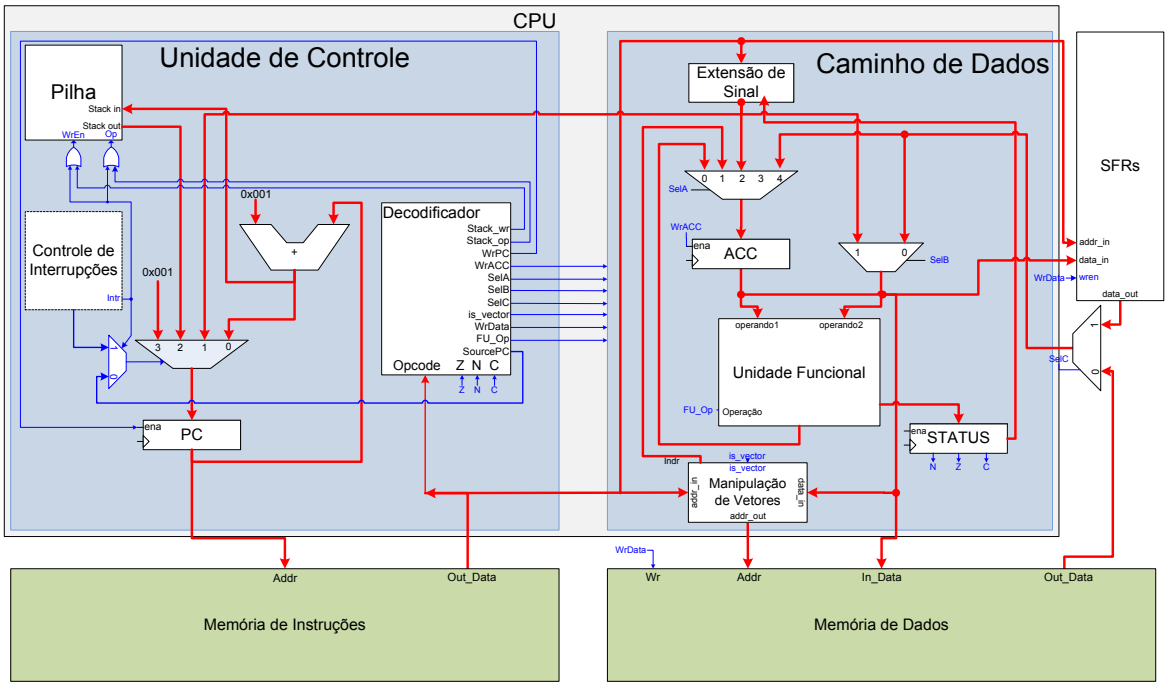


Figura 6.10. Organização do μBIP

Tabela 6.11. Conjunto de instruções estendido pelo μBIP

Código da Operação	Instrução	Operação	Classe
11000	STOV operando	Memória[operando + INDR] ← ACC	Manipulação de vetor
11001	LDV operando	ACC ← Memória[operando + INDR]	Manipulação de vetor
11010	RETURN	PC ← ToS	Suporte a procedimentos
11011	RETINT	PC ← ToS	Suporte a procedimentos
11100	CALL operando	PC ← operando ToS ← PC+1	Suporte a procedimentos

O μBIP foi desenvolvido utilizando-se o ArchC [The ArchC Team 2007] como suporte para especificação arquitetural e geração de código. O ArchC é uma linguagem de descrição de arquitetura (Architecture Description Language, ADL) baseada no SystemC e que foi desenvolvida pelo Instituto de Computação da Universidade de Campinas (IC-UNICAMP). O ArchC permite descrever a arquitetura do processador e a sua hierarquia de memória, gerando, automaticamente, ferramentas como um simulador de conjunto de instruções, um montador, um ligador e um depurador, as quais foram geradas para o μBIP. A especificação arquitetural foi validada com base em testes unitários das instruções e na execução de aplicações especificadas no testbench do Dalton Project [The Dalton Project Team 2001].

Além das ferramentas geradas com o auxílio do ArchC, no âmbito do trabalho de conclusão de curso no qual o μBIP foi desenvolvido, também foi implementado um modelo do microcontrolador na linguagem de descrição de hardware VHDL, o qual foi sintetizado e prototipado em FPGA (Field Programmable Gate Array). O protótipo físico foi validado executando-se as aplicações do Dalton Project.

O uso do BIP como referência para este trabalho mostrou que, apesar de suas limitações arquiteturais, definidas de modo a favorecer o aprendizado, a arquitetura do BIP pode ser estendida para incluir funcionalidades tipicamente encontradas em processadores comerciais.

O μBIP ainda não foi aplicado em atividades de ensino. Por enquanto, estão sendo realizadas atividades de desenvolvimento no sentido de integrar periféricos de comunicação serial ao microcontrolador. No futuro, pretende-se utilizar a experiência obtida nesses desenvolvimentos para elaborar roteiros de projetos a serem realizados em disciplinas avançadas na área de projeto de sistemas computacionais, conforme já mencionado.

6.3.6.2 BipIde

Neste trabalho de conclusão de curso [Viera 2009][Vieira, Raabe e Zeferino 2010], foi implementada uma IDE para desenvolvimento de pequenos algoritmos em Portugol e sua conversão na linguagem de montagem do BIP. A IDE inclui um simulador da arquitetura e da organização do BIP II que possibilita aos alunos testarem seus programas, acompanhando as ações realizadas no nível da organização do processador. A Figura 6.11 (mostrada na próxima página) ilustra a interface do BipIde, cujos recursos são descritos a seguir.

1. Simulação: apresenta os botões que permitem ao usuário controlar a simulação do programa e verificar sua execução passo-a-passo, incluindo as ações a seguir:
    - Simular: inicia a simulação de um programa;
    - Pausar: interrompe temporariamente a simulação do programa;
    - Parar: interrompe a simulação do programa;
    - Continuar: continua a simulação do programa até o seu final;
    - Repetir: repete a última instrução simulada; e
    - Próximo: simula a próxima instrução do programa.
2. Velocidade: através desta opção, o usuário pode controlar a velocidade com que ocorre a simulação do programa;
3. Simulador de instruções: o usuário pode simular uma instrução específica do processador, sem a necessidade de escrever um programa completo. Para isso, poderá ser digitado o valor do operando no campo específico e pressionado o botão correspondente à instrução que deverá ser simulada;
4. Portugol: exibe o programa Portugol que está sendo simulado. Neste editor, a linha do programa que está sendo simulada aparece destacada;

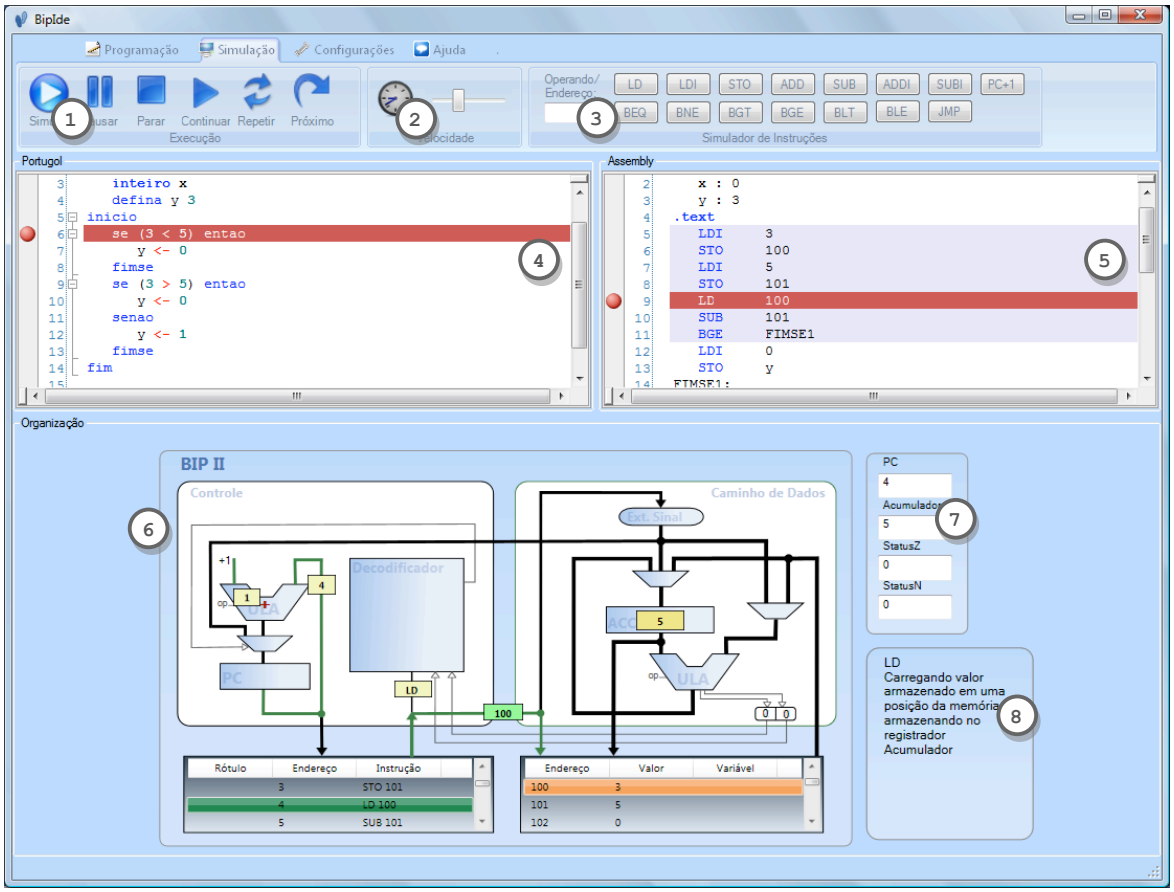


Figura 6.11. Interface do BipIde

5. Assembly: exibe o código *assembly* gerado pelo compilador. Neste editor, a linha do programa que está sendo simulada aparece destacada. É destacado também o bloco de instruções *assembly* que correspondem à linha do código em alto nível destacada na janela Portugol, permitindo ao usuário verificar quais instruções *assembly* foram geradas para executar uma instrução Portugol. No programa da Figura 6.11, foram geradas sete linhas de código *assembly*, para executar a instrução Portugol correspondente em destaque;
6. Organização do processador: exibe a imagem da organização do processador (BIP I ou BIP II), sobre a qual são feitas animações que representam a execução das instruções no processador. A fim de facilitar o entendimento do simulador, os seus principais componentes são nomeados e, com dois cliques sobre eles, é exibida uma janela informativa a respeito do componente;
7. Registradores: exibe os valores dos registradores do processador durante a simulação do programa; e
8. Descrição: exibe o nome da instrução simulada e uma descrição da ação ocorrida, a fim de auxiliar o usuário a compreender o que está ocorrendo no processador.

A ferramenta tem sido utilizada para apoiar a apresentação do BIP I e BIP II na disciplina Algoritmos e Programação do primeiro período e também na disciplina Compiladores para fornecer um ambiente de simulação para o código gerado pelos compiladores desenvolvidos pelos alunos.

6.4 Considerações finais

A abordagem apresentada neste capítulo enfatizou uma visão multi e interdisciplinar do ensino de conceitos de arquitetura de computadores, sendo que o desenvolvimento da família de processadores levou em conta as necessidades das diferentes disciplinas envolvidas. Essa característica configura-se como principal diferencial desta proposta.

As atividades de aprendizagem já conduzidas apontaram diversos aspectos positivos. Os principais são:

1. Uma integração mais natural entre os conceitos de programação, circuitos digitais, arquitetura de computadores e compiladores;
2. A possibilidade de utilizar um mesmo modelo de computação durante boa parte do percurso do aluno na graduação;
3. A redução de problemas de aprendizagem relacionados com as abstrações envolvidas nos conceitos introdutórios de programação, tais como: variáveis, constantes e atribuições;
4. Uma ilustração melhor fundamentada aos alunos dos períodos iniciais da relação entre os desvios, laços, a linguagem de montagem e o funcionamento do processador;
5. Facilidade na consolidação de conceitos de arquitetura e organização de computadores pelo uso de uma arquitetura de referência em diferentes disciplinas;
6. Consolidação dos conceitos de circuitos digitais pelo uso do BIP como projeto integrador ao final da disciplina;
7. Ao utilizar o conjunto de instruções do BIP, a etapa de geração de código dos trabalhos de compiladores foi concluída pela maioria dos alunos;
8. Após a experiência no uso do BIP, iniciou-se um processo de reflexão acerca do melhor encadeamento dos conteúdos entre as disciplinas envolvidas. Como resultado dessa reflexão, foram identificados aspectos desejáveis que serão buscados em trabalhos futuros:
    - Rever a organização curricular do curso buscando reduzir a distância entre a disciplina Arquitetura de Organização de Computadores e a disciplina Compiladores;
    - Promover o uso dos processadores BIP em outras disciplinas, como, por exemplo, a disciplina de Sistemas Operacionais; e
    - Conduzir experimentos controlados a fim de obter evidências empíricas sobre a redução dos problemas de aprendizagem associados à necessidade de abstração.

6.5 Agradecimentos

Os autores agradecem ao CNPq pelo apoio concedido para realização da pesquisa via Edital Universal 2008 (Processo No. 477820/2008-5), à Capes pelo apoio à continuidade da pesquisa via Edital Prosup 2010 – Cursos Novos (Auxílio No. 2140/2010) e à UNIVALI pelo contínuo suporte às atividades desta pesquisa.

6.6 Referências

Carbone, A. e Kaasboll, J. (1998) "A survey of methods used to evaluate computer science teaching". In: Proceedings of the 3rd Conference on Teaching of Computing, Dublin, p. 41-45.

Clements, A. (1999) "Selecting a processor for teaching computer architecture", Microprocessor and Microsystems, v.23, n.5, p. 281-290.

Good, J. e Brna, P. (2004) "Program comprehension and authentic measurement: a scheme for analyzing descriptions of programs", Journal of Human-Computer Studies, v.61, n.2, p. 169-185.

Hostins, H. e Raabe, A. L. A. "Auxiliando a aprendizagem de algoritmos com a ferramenta Webportugol". In: XIV Workshop de Educação em Computação – XXVII Congresso da SBC, 2007, Rio de Janeiro. Anais do XXVII Congresso da SBC, 2007. v. 1. p. 96-105.

Khalife, J. T. (2006) "Threshold for the introduction of Programming: Providing Learners with a Simple Computer Model" In: Proceedings of the 28th International Conference on Information Technology Interfaces, p. 71-76.

Lins, R. D. (2000) "Uma Proposta de Plano Pedagógico para a matéria de Compiladores". In: II Curso de Qualidade de Cursos de Graduação da Área de Computação e Informática, Congresso Anual da SBC.

Menezes, C. S. e Nobre, A. M. (2002) "Um ambiente cooperativo para apoio a cursos de introdução a programação". In: Workshop de Educação em Computação, Anais do XXII Congresso da Sociedade Brasileira de Computação. Porto Alegre: SBC.

Microchip (2003) PIC16F62X Data Sheet: FLASH-Based 8-Bit CMOS Microcontroller. Arizona.

Morandi, D., Raabe, A. L. A. e Zeferino, C. A. (2006) "Processadores para ensino de conceitos básicos de Arquitetura de Computadores" In: Anais do 1º Workshop sobre Educação em Arquitetura de Computadores, p. 17-24.

Morandi, D., Pereira, M. C., Raabe, A. L. A. e Zeferino, C. A. (2006) "Um processador básico para o ensino de conceitos de arquitetura e organização de computadores". Revista Hífen, v. 30, n. 58, p. 73-80.

Parr, T. (2009) ANTLR v3 Documentation. Disponível em: <http://www.antlr.org>.

Patterson, D. A. e Hennessy, J. L. (2005) Organização e projeto de computadores: a interface hardware/software, São Paulo, Campus.

Pereira, M. C. e Zeferino, C. A. (2008) “uBIP: a simplified microcontroller architecture for education in embedded systems design. In: Proceedings of the IP Based

Electronic System Conference & Exhibition - IP 08, Grenoble : Design and Reuse, p. 193-197.

Pereira, M. C. (2008) BIP: Microcontrolador Básico para o Ensino de Sistemas Embarcados. Trabalho de Conclusão de Curso, Ciência da Computação, Universidade do Vale do Itajaí.

Pimentel, E. P., Franca, V. F., Noronha, R. V. e Omar, N. (2003) "Avaliação contínua da aprendizagem, das competências e habilidades em programação de computadores". In: Workshop de Informática na Escola, Anais do XXIII Congresso da Sociedade Brasileira de Computação. Porto Alegre: SBC.

The ArchC Team (2007) The ArchC project home page, Disponível em: <http://www.archc.org >.

The Dalton Project Team (2001) The UCR Dalton Project, University of California, Riverside. Disponível em: < http://www.cs.ucr.edu/~dalton/ >.

Vieira, P. V. (2009) BipIde: Ambiente de Desenvolvimento Integrado para a Arquitetura dos Processadores BIP. Trabalho de Conclusão de Curso, Ciência da Computação, Universidade do Vale do Itajaí.

Vieira, P. V., Raabe, A. L. A. e Zeferino, C. A. (2010) Bipide: ambiente de desenvolvimento integrado para a arquitetura dos processadores BIP. Revista Brasileira de Informática na Educação, v. 18, p. 32-43.

Weber, R. F. (2004) Fundamentos de arquitetura de computadores, Porto Alegre, Sagra Luzzatto.

Capítulo

7

O Termo Arquitetura de Computador no Passado, Presente e Futuro: impactos do seu significado no ensino e no aprendizado de Arquitetura de Computadores

Carlos Augusto Paiva da Silva Martins

Resumo

Neste capítulo são indicadas e analisadas algumas das principais definições e significados do termo "Arquitetura de Computador" que foram apresentadas na literatura ao longo dos anos. São analisados os principais impactos do significado do termo nos processos de ensino e de aprendizado de arquitetura de computadores no Brasil. É apresentada uma proposta de definição para o termo que considera de modo integrado e hierárquico as arquiteturas do sistema de computação, do software e do hardware. São apresentadas sugestões de atividades relacionadas ao ensino e aprendizado de arquitetura de computadores que utilizam as definições e significados do termo de modo motivador e citados alguns resultados iniciais produzidos com a sua utilização. Finalmente, apresenta-se a conclusão que possui uma discussão dos ganhos produzidos pela utilização motivadora do termo e do seu significado. Espera-se que esse capítulo contribua nos aumentos de interesse, motivação e qualidade no aprendizado da arquitetura de computadores. Ao longo do capítulo utilizam-se perguntas para motivar estudos, análises, reflexões, discussões e pesquisas.

7.1. Introdução

No contexto geral da arquitetura de computadores, um grande número, ou até mesmo a maioria, dos alunos dos cursos da área de computação é pouco interessado e ou pouco motivado para o aprendizado dos tópicos relacionados com a arquitetura de computadores. Este fato é causado, pelo menos em parte, pela crença e entendimento que a arquitetura de computadores está relacionada somente com os aspectos do hardware dos computadores. A maior parte das definições do termo "arquitetura de computador", utilizadas nos últimos anos e atualmente, contribui para este entendimento e também para a produção desses problemas [Hennessy 2012].

Além disso, muitos professores, principalmente das disciplinas que são realizadas no início dos cursos e ou que não são relacionadas diretamente com a área de arquitetura de computadores, também não possuem grande interesse nos seus tópicos, nem mesmo de modo aplicado e não valorizam a utilização dos conhecimentos da área de arquitetura de computadores nas suas áreas de atuação e especialidades. Deste modo, esses professores não apresentam e não exemplificam para os alunos a importância dos tópicos de arquitetura de computadores para as atividades relacionadas com o projeto e o desenvolvimento de técnicas, algoritmos e softwares nas suas respectivas disciplinas e principalmente na formação dos profissionais de todas as demais áreas relacionadas à computação.

Os fatos apresentados anteriormente, pelo menos indiretamente, contribuem muito para o desinteresse e a desmotivação dos alunos em relação ao estudo e ao aprendizado dos tópicos relacionados com a arquitetura de computadores. Nesse contexto geral da educação em computação, existem inúmeras evidências que esses problemas são, pelo menos em parte, causados pelas definições e pelos significados mais difundidos do termo "arquitetura de computador", e por suas influências nos tópicos que são ensinados e aprendidos nas disciplinas diretamente relacionadas com a arquitetura de computadores e também por suas influências nos aspectos metodológicos utilizados nos processos de ensino e aprendizagem de arquitetura de computadores.

Deste modo, os problemas motivadores da proposta e da escrita deste capítulo são os efeitos e os impactos negativos na educação em computação, de modo geral, e especialmente na educação em arquitetura de computadores, causados pelas definições e pelos significados do termo arquitetura de computador que são utilizados e considerados.

A principal consequência prática desses problemas motivadores é a formação de egressos dos cursos da área de computação com conhecimentos insuficientes e, em alguns casos, até deficientes nos tópicos relacionados com a arquitetura de computadores. Isso produz uma falta de recursos humanos especializados na área de arquitetura de computadores, observada no Brasil principalmente nas atividades de docência e de pesquisa. E de modo mais impactante, a formação de profissionais egressos dos diversos cursos da área de computação com muitas dificuldades para o projeto e o desenvolvimento de algoritmos e de softwares que utilizem os sistemas de computação de modo mais eficiente e otimizado em relação às suas características arquiteturais.

A motivação principal para a escrita deste capítulo foi produzir um texto em língua portuguesa relacionado à educação em arquitetura de computadores, que possa ser utilizado por alunos e professores, apresentando e analisando:

- Algumas das principais definições do termo “arquitetura de computador” que foram apresentadas na literatura ao longo dos anos, considerando os contextos da criação, evolução e atualidade do termo arquitetura de computador;

- Os principais impactos positivos e negativos das definições e dos significados do termo “arquitetura de computador” nos processos de ensino e de aprendizado dos conteúdos da própria arquitetura de computadores e também, de modo geral, nos processos de ensino e de aprendizado da computação, nos diversos cursos relacionados com a área de computação;

- Algumas sugestões de atividades relacionadas ao ensino e aprendizado de arquitetura de computadores que utilizam as definições e significados do termo de modo motivador. São citados alguns resultados iniciais produzidos com a sua utilização;

- Uma proposta, versão inicial, de nova definição para o termo “arquitetura de computador”, que considera de modo integrado e hierárquico as arquiteturas do sistema de computação, do software e do hardware. Esta proposta é motivada pelos possíveis impactos positivos dessa redefinição, com o propósito de iniciar uma discussão sobre a importância, a relevância e a necessidade da redefinição do termo “Arquitetura de Computador”. Em função das dificuldades e da complexidade dessa tarefa, acredita-se que esta redefinição deve ser um processo coletivo (cooperativo e colaborativo) envolvendo inicialmente os representantes das comunidades acadêmica, científica e industrial que atuam no contexto da arquitetura de computadores e posteriormente os representantes das comunidades acadêmica, científica e industrial que atuam no contexto mais amplo da computação;

- Uma discussão dos possíveis ganhos produzidos pela utilização motivadora do termo e do seu significado na educação em arquitetura de computadores.

Espera-se que esse capítulo seja um material de referência e que a utilização do seu conteúdo possa auxiliar e contribuir no aumento do interesse, motivação e dedicação dos alunos para o estudo e o aprendizado de alta qualidade dos tópicos relacionados com a arquitetura de computadores. Espera-se também que uma das principais causas desses aumentos seja o entendimento, de modo teórico e aplicado, que a arquitetura de computadores não é relacionada somente com o hardware e que está relacionada com os computadores ou com os sistemas de computação de modo sistêmico (hierárquico), considerando os diversos níveis hierárquicos, dos módulos de software e dos módulos de hardware de modo isolado e principalmente de modo integrado. Deste modo, os alunos perceberão, na teoria e principalmente na prática, que os conhecimentos relacionados com a arquitetura de computadores são importantes e essenciais também para os profissionais envolvidos somente ou principalmente com o projeto e o desenvolvimento de software (aplicativo e de sistema), que sempre possui requisitos relacionados à utilização otimizada e eficiente dos recursos dos computadores.

7.2. O termo “Arquitetura de Computador”

Nesta seção, apresentam-se os termos utilizados antes da origem do termo arquitetura de computador, analisa-se o seu significado original e posteriormente analisa-se a evolução do termo e do seu significado ao longo dos anos.

7.2.1. Os termos utilizados antes do termo “Arquitetura de Computador”

Uma análise dos documentos (artigos, manuais, relatórios técnicos, etc) relacionados aos primeiros computadores eletrônicos digitais projetados e implementados, mostra que o termo *“Computer Architecture”* não era utilizado e que os termos mais utilizados eram *“Computer Structure”* e *“Computer Organization”*.

Dentre os primeiros computadores eletrônicos digitais, pelas suas importâncias históricas, foram selecionados e analisados o *Electronic Numerical Integrator and Computer* (ENIAC), *Electronic Discrete Variable Automatic Computer* (EDVAC), *Electronic Delay Storage Automatic Calculator (*EDSAC), *Universal Automatic Computer I* (UNIVAC I).

No contexto do ENIAC, primeiro computador eletrônico digital de propósito geral, foram analisados os documentos *“A Report on the ENIAC”* [Goldstine 1946], *“The Electronic Numerical Integrator and Computer (ENIAC)”* [Goldstine 1946], *“Electronic Computing Circuits of the ENIAC”* [Burks 1947], *"The ENIAC"* [Brainerd 1948] e *“The ENIAC: The First General-Purpose Electronic Computer”* [Burks 1981].

No contexto do EDVAC, primeiro computador eletrônico digital com programa armazenado, foram analisados os documentos *“First Draft of a Report on the EDVAC"* [Von Neumann 1945] e *“The Origins, Uses, and Fate of the EDVAC”* [Williams 1993] e *“The Computer as von Neumann Planned It”* [Godfrey 1993].

No contexto do EDSAC, segundo computador eletrônico digital com programa armazenado, foram analisados os documentos *“The EDSAC Computer"* [Wilkes 1951] e *"Early computer developments at Cambridge: The EDSAC"* [Wilkes 1975].

No contexto do UNIVAC, primeiro computador eletrônico digital produzido comercialmente nos Estados Unidos, foram analisados os documentos *“The UNIVAC System”* [Eckert 1951] e *“From ENIAC to UNIVAC, An appraisal of the Eckert-Mauchly Computers”* [Stern 1981]

Neste contexto inicial, como é analisado com mais detalhes na próxima subseção, a primeira utilização do termo *“Computer Architecture”* em um documento público foi no livro *“Planning a Computer System: Project Stretch”* [Buchholz 1962].

7.2.2. Origem do termo Arquitetura de Computador

Atualmente, acredita-se que o termo “Arquitetura de Computador” foi criado originalmente no idioma inglês *“Computer Architecture”* e definido como *“Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints. Architecture must include engineering considerations, so that the design will be economica1 and feasible; but the emphasis in architecture is upon the needs of the user, whereas in engineering the emphasis is upon*

the needs of the fabricator." e publicado inicialmente no capítulo 2, intitulado "Architectural Philosophy" escrito por Frederick Phillips Brooks Jr., do livro intitulado "Planning a Computer System: Project Stretch" editado por Werner Buchholz [Buchholz 1962].

Recomenda-se fortemente a leitura desse capítulo 2, enfatizando-se que outros termos também são utilizados ao longo dos capítulos do livro. Destaque especial para o capítulo 3 do mesmo livro.

Entretanto, por muitos anos acreditou-se que a definição original do termo era *"The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation"* que foi apresentada no artigo intitulado *"Architecture of the IBM System/360"*, escrito por G.M. Amdahl, G.A. Blaauw e F. P. Brooks Jr. e publicado no IBM Journal of Research and Development [Amdahl 1964].

7.2.3. Evolução do termo Arquitetura de Computador

Ao longo dos anos, a definição original do termo "Computer Architecture" foi alterada ou modificada por diversos autores e apresentada em inúmeras fontes da literatura, mas essencialmente permaneceu a ideia, entendimento ou significado que a arquitetura dos computadores é relacionada exclusivamente com o hardware do computador.

Uma prova desse fato, é a definição "*Computer Architecture is the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals. Computer architecture is not about using computers to design buildings."* atualmente apresentada no principal site relacionado com a arquitetura de computadores que é o "WWW Computer Architecture Page" []

Na atualidade, dentre as diversas definições existentes para o termo "Computer Architecture", destacam-se as seguintes:
- A definição apresentada na quinta edição do livro "Computer Architecture: a quantitative approach" que é a seguinte "In this book, the word architecture covers all three aspects of computer design – instruction set architecture, organization or mocroarchitecture, and hardware." [Hennessy 2012]. Neste livro, são apresentados os seguintes trechos: "Several years ago, the term computer architecture often referres only to instruction set design. Other aspects of computer design were called implementation, often insinuating that implementation is uninteresting or less challenging."

"We believe this view is incorrect. The architect's or designer's job is much more than instruction set design, and the technical hurdles in the other aspects of the project are likely more challenging than those encoutered in instruction set design." [Hennessy 2012]

Nos contextos da origem e da atualidade das definições do termo, pode-se e deve-se refletir sobre algumas perguntas e principalmente sobre algumas das suas possíveis respostas:

1) A arquitetura de um computador está relacionada somente com os componentes do seu hardware?

2) Os componentes do *firmware*, não estão relacionados com a arquitetura do computador?

3) Os componentes do software básico, como o sistema operacional e outros, não estão relacionados com a arquitetura do computador?

4) Esta definição é independente do tipo e da complexidade do computador considerado, como por exemplo um computador embutido, um tablet, um desktop, um servidor ou um supercomputador?

5) Existe alguma diferença entre a "Arquitetura do computador" e a "Arquitetura do hardware do computador" ou são a mesma coisa? Se existem alguma diferença, então a arquitetura de um computador não está relacionada somente com o seu hardware.

6) Durante a execução de um software aplicativo em um determinado computador, a arquitetura desse computador pode influenciar o tempo de resposta, vazão e outras métricas relacionadas ao desempenho computacional?

Finalmente, destaca-se que nas tabelas das áreas do conhecimento, a área da computação possui uma subárea denominada "Sistemas de Computação" e dentro dessa última existem especialidades denominadas "Arquitetura de Computadores" e "Hardware".

Neste contexto, de evolução do termo "Arquitetura de Computadores" ao longo dos anos, outros termos muito relacionados como "Estrutura de Computador" e "Organização de Computador" continuam sendo utilizados, porém com menor enfase e destaque. Além disso, também surgiram e ou evoluíram outros termos relacionados como "Arquitetura de Hardware", "Arquitetura de Software" e "Arquitetura de Sistema". Entretanto, não encontramos definições de destaque na literatura para o termo "Arquitetura de Computador" que utilizem diretamente esses últimos três termos relacionados.

Deste modo, analisando-se as diversas definições do termo utilizadas na atualidade e as possíveis respostas para as perguntas anteriores, conclui-se que uma nova definição do termo "computer architecture", mais atualizada e abrangente, poderia produzir impactos positivos de modo particular na educação em arquitetura de computadores e de modo geral na própria área da arquitetura de computadores. O objeto denominado computador ou sistema de computação, tem evoluído muito ao longo dos anos e as suas possíveis arquiteturas também tem evoluído muito. Na atualidade, com o domínio dos computadores com arquiteturas paralelas, observa-se um aumento na diversidade e na heterogeneidade das arquiteturas dos sistemas de computação.

7.2.4. O futuro do termo Arquitetura de Computador

Em relação ao futuro do termo, deve-se considerar o futuro da computação com um todo, principalmente os diferentes modelos de computação, inclusive os híbridos e não convencionais, o futuro dos computadores, principalmente em relação às diferentes arquiteturas de computadores e às diferentes tecnologias de implementação de sistemas

de computação. Neste contexto futuro, deve-se considerar os computadores de propósito geral e os computadores de propósito específico e ou dedicado.

Neste contexto do futuro do termo, as seguintes peguntas são fundamentais:

- Quais são as principais tendências e perspectivas futuras de utilização dos diversos tipos de sistemas de computação?

- Quais são as principais tendências e perspectivas futuras em termos de arquiteturas dos diversos tipos de sistemas de computação?

- Quais são as principais tendências e perspectivas futuras em termos de tecnologias de implementação dos diversos tipos de sistemas de computação?

Considera-se que no futuro, em função das principais tendências futuras da computação e dos diversos tipos de sistemas de computação, incluindo os denominados não convencionais ou não tradicionais, sendo que em alguns desses sistemas não existe software e também não existe arquitetura de software, os impactos negativos produzidos pelas definições e significados do termo arquitetura de computadores poderão ser ainda maiores.

Deste modo, devemos analisar quais são as principais tendências e perspectivas futuras de utilização e de significado do termo, considerando os diversos tipos de sistemas de computação incluindo os denominados não convencionais ou não tradicionais, sendo que em alguns desses sistemas não existe software e também não existe arquitetura de software.

Acredita-se que as definições do termo devam ser abrangentes, robustas e pouco variantes no tempo. Deste modo, devem existir definições mais abstratas, que serão as mais robustas e invariantes, e definições menos abstratas ou mais concretas que serão as menos robustas e mais variantes entre as diferentes arquiteturas de computadores e também ao longo do tempo.

7.3. Impactos do significado do termo na educação em arquitetura de computadores

Nesta seção apresentam-se e analisam-se, exemplificando-se sempre que possível, os principais impactos positivos e negativos dos significados do termo “Arquitetura de Computador” no processo de educação em arquitetura de computadores, considerando-se de modos isolado e integrado o ensino e o aprendizado.

São considerados isoladamente as influências e os impactos causados, entre os quais destacam-se os relacionados ao interesse, a motivação, a dedicação e a facilidade de entendimento dos alunos das disciplinas diretamente relacionadas com arquitetura de computadores, dos professores das disciplinas relacionadas com arquitetura de computadores e dos alunos e professores das demais disciplinas.

Ao analisar-se os impactos do termo na educação em arquitetura de computadores, deve-se considerar individualmente e diferentemente os cursos de ciência da computação e os demais cursos relacionados com a computação. Isso deve-se aos diferentes perfis dos egressos dos diversos cursos da área e à quantidade e profundidade dos conteúdos relacionados com a arquitetura de computadores ensinados pelos professores e aprendidos pelos alunos dos diferentes cursos. Finalmente, observa-se que também existem diferenças entre cursos com mesma denominação de diferentes instituições de ensino, em função dos projetos pedagógicos dos cursos, da formação dos professores e dos perfis esperados dos egressos de cada instituição.

Nesta seção, analisam-se os impactos do termo na educação em arquitetura de computadores somente nos cursos de ciência da computação, em função da experiência do autor. Espera-se no futuro, com a participação de outros professores, realizar análises semelhantes nos demais cursos da área da computação.

Nas análises considera-se a educação em arquitetura de computadores tanto no contexto das disciplinas denominadas ou diretamente relacionadas com arquitetura de computadores quanto nas disciplinas não relacionadas diretamente, mas que muitas vezes são as ideais para a apresentação informativa e aprendizado de tópicos e ou para a valorização e a motivação inicial para o aprendizado dos tópicos relacionados com a arquitetura de computadores. Deste modo, existem impactos nas próprias disciplinas diretamente relacionadas com a arquitetura de computadores e também nas demais disciplinas dos cursos da área de computação.

Além disso, existem impactos negativos de longo prazo, pois os futuros professores das demais disciplinas dos diversos cursos da área de computação, normalmente são formados nos cursos de graduação com a crença e o entendimento que a arquitetura de computadores está relacionada exclusivamente com o hardware e que não é muito importante ou essencial para os alunos interessados no desenvolvimento de algoritmos e softwares.

Nas análises dos impactos, deve-se considerar que todos os softwares, para produzir os seus resultados desejados e esperados, serão executados em sistemas de computação implementados fisicamente e que as características arquiteturais influenciam diretamente na qualidade dos resultados e no desempenho computacional. Uma análise

mais profunda, mostra que as características arquiteturais influenciam diretamente no alcance dos requisitos funcionais e principalmente no alcance dos requisitos não funcionais desses softwares. Deste modo, não seria absurdo considerar que o aprendizado de modo aplicado dos conhecimentos de arquitetura de computadores são essenciais e talvez estejam entre os mais importantes para todos os profissionais da área de computação, independentemente da sua área de especialização.

7.3.1. Impactos do significado do termo nos períodos anteriores às disciplinas Arquitetura de Computadores

Nas disciplinas iniciais dos cursos, principalmente nas relacionadas com algoritmos, estruturas de dados e programação, observa-se que na maior parte dos cursos os alunos não são motivados pelos seus professores a considerar e valorizar os aspectos relacionados com as arquiteturas e com as implementações dos computadores que são utilizados para a execução dos programas desenvolvidos a partir dos algoritmos propostos e elaborados. Normalmente, as influências das arquiteturas no desempenho computacional e até mesmo na qualidade dos resultados não são apresentadas e discutidas pelos professores e posteriormente exploradas e investigadas pelos alunos.

Deve-se observar que a maior parte desses alunos possuem computadores com processadores de propósito geral com múltiplos núcleos e que os impactos das arquiteturas nos desempenhos dos algoritmos e softwares desenvolvidos poderiam ser explorados de modo informativo e motivador ao aprendizado de arquitetura de computadores.

Considerando-se os períodos anteriores à realização das disciplinas diretamente relacionadas com a arquitetura de computadores, ou os períodos iniciais dos cursos, observa-se que os alunos não possuem ou possuem poucos conhecimentos relacionados com a arquitetura de computadores, inclusive sobre as definições do termo. Deste modo, observa-se que os principais impactos negativos relacionados às definições e significados do termo “Arquitetura de Computadores” são causados pelas opiniões, crenças e ações dos professores das disciplinas cursadas pelos alunos.

Os atuais professores, atuam nas suas atividades docentes influenciados pelos conhecimentos aprendidos e pelas crenças inicialmente construídas na época em que eram alunos. Deste modo, observa-se uma maior dificuldade para alterar o significado do termo e minimizar os seus impactos na qualidade do aprendizado dos alunos dos períodos anteriores, pois inicialmente deve-se alterar o significado do termo e a conduta dos seus professores das demais disciplinas.

Estes fatos, indicam a necessidade de desenvolver-se atividades específicas para e com os professores das demais disciplinas e especialidades da computação, no curto prazo, para minimizar os impactos negativos e inicialmente aumentar o interesse e a motivação dos professores no aprendizado e uso aplicado dos conhecimentos de arquitetura de computadores nas suas especialidades, para posteriormente minimizar os impactos negativos sobre os alunos durante essas disciplinas.

Entretanto, ações e atividades podem ser propostas pelos professores de arquitetura de computadores aos professores dessas disciplinas anteriores. Na seção 7.5, algumas dessas atividades são indicadas.

7.3.2. Impactos do significado do termo nas disciplinas diretamente relacionadas à Arquitetura de Computadores

Nas disciplinas diretamente relacionadas à arquitetura de computadores, observa-se que normalmente os professores apresentam e os alunos aprendem os tópicos ou conhecimentos de arquitetura de computadores. Entretanto, observa-se que um menor número de professores apresentam e de alunos aprendem métodos e técnicas de projeto e de tomada de decisão em contextos projeto e ou de utilização prática dos conhecimentos de arquitetura de computadores. Finalmente, em poucos cursos os alunos são motivados pelos seus professores a utilizar os aspectos e conhecimentos relacionados com arquitetura dos computadores para a melhoria do desempenho computacional de diferentes tipos de algoritmos e programas que devem ser propostos e elaborados com este propósito. Normalmente, as influências dos modelos de computação (sequencial, paralelo, distribuído, etc) e das arquiteturas dos computadores no desempenho computacional e até mesmo na qualidade dos resultados não são apresentadas e discutidas pelos professores e posteriormente exploradas e investigadas de modo mais profundo pelos alunos.

Estes fatos, são muito graves, pois a maior parte dos alunos que iniciam as disciplinas diretamente relacionadas com arquitetura de computadores é pouco interessada e ou pouco motivada para o aprendizado dos tópicos relacionados com a arquitetura de computadores. E as atividades que integram o uso de tópicos de arquitetura no desenvolvimento de programas com melhores requisitos funcionais e principalmente com melhores requisitos não funcionais podem ser um dos caminhos mais eficientes para motivar e valorizar o entendimento e o aprendizado de arquitetura de computadores. Deste modo, os alunos percebem que a arquitetura de computadores não está relacionada somente com os aspectos do hardware dos computadores.

Finalmente, as atividades desenvolvidas podem ser complementadas com a apresentação de novas definições do termo “arquitetura de computador”, mostrando que o termo é relacionado com aspectos sistêmicos, do software e do hardware do computador. E também que os conhecimentos de arquitetura são essenciais para o projeto e o desenvolvimentos de algoritmos e softwares de melhor desempenho e qualidade.

Finalmente, observa-se que muito, senão a maioria, dos professores que lecionam as disciplinas diretamente relacionadas com Arquitetura de Computadores não são especialistas, não realizam pesquisas na área e possuem formação semelhante a muitos professores das demais disciplinas. Este fato, é muito grave, principalmente pelo risco de prejudicar simultaneamente os aspectos de motivação e qualidade do aprendizado.

Existem impactos negativos nos tópicos e conteúdos que normalmente são ensinados e aprendidos nas disciplinas diretamente relacionadas com a arquitetura de computadores, mas acredita-se que os maiores impactos negativos estão relacionados com os aspectos metodológicos utilizados nessas disciplinas. Além disso, um outro impacto negativo é o aprendizado de arquitetura de computadores de modo pouco integrado com o desenvolvimento de software aplicativo e ou básico.

7.3.3. Impactos do significado do termo nos períodos posteriores às disciplinas Arquitetura de Computadores

Nas disciplinas dos períodos posteriores às disciplinas Arquitetura de Computadores, principalmente nas relacionadas com engenharia de software e com as principais áreas de aplicação, observa-se que na maior parte dos cursos os alunos não são motivados pelos seus professores a continuar considerando e valorizando os aspectos relacionados com as arquiteturas e com as implementações dos computadores que são utilizados para a execução dos programas desenvolvidos a partir dos algoritmos propostos e elaborados. Normalmente, as influências das arquiteturas no desempenho computacional e até mesmo na qualidade dos resultados não são apresentadas e discutidas pelos professores e posteriormente exploradas e investigadas pelos alunos.

Considerando-se os períodos posteriores à realização das disciplinas diretamente relacionadas com a arquitetura de computadores, ou os períodos finais dos cursos, observa-se que os alunos já possuem muitos conhecimentos relacionados com a arquitetura de computadores, inclusive sobre as definições do termo. Deste modo, observa-se que os principais impactos negativos relacionados às definições e significados do termo “Arquitetura de Computadores” são causados pelas opiniões, crenças e ações dos próprios alunos e também são influenciadas pelas opiniões, crenças e ações dos professores das disciplinas cursadas pelos alunos.

Os atuais professores, atuam nas suas atividades docentes influenciados pelos conhecimentos aprendidos e pelas crenças inicialmente construídas na época em que eram alunos. Deste modo, observa-se uma maior dificuldade para alterar o significado do termo e minimizar os seus impactos na qualidade do aprendizado dos alunos dos períodos anteriores, pois inicialmente deve-se alterar o significado do termo e a conduta dos seus professores das demais disciplinas. Entretanto, observa-se uma menor dificuldade para alterar o significado do termo e minimizar os seus impactos na qualidade do aprendizado dos alunos dos períodos posteriores, pois os alunos já possuem conhecimentos teóricos de arquitetura de computadores e aplicados ao desenvolvimento de algoritmos e programas otimizados pela utilização de conhecimentos de arquitetura de computadores. Mesmo assim, deve-se alterar o significado do termo e a conduta dos professores dessas disciplinas posteriores.

Estes fatos, indicam a necessidade de desenvolver-se atividades específicas para e com os professores das demais disciplinas e especialidades da computação, no curto prazo, para minimizar os impactos negativos e inicialmente aumentar o interesse e a motivação dos professores no aprendizado e uso aplicado dos conhecimentos de arquitetura de computadores nas suas especialidades, para posteriormente minimizar os impactos negativos sobre os alunos durante essas disciplinas.

Entretanto, ações e atividades podem ser propostas pelos professores de arquitetura de computadores aos professores dessas disciplinas posteriores. Na seção 7.5, algumas dessas atividades são indicadas.

Na próxima seção, é apresentada uma proposta de (re)definição para o termo arquitetura de computadores que poderia produzir mais impactos positivos e menos impactos negativos no processo de educação em arquitetura de computadores e de modo global no processo de educação em computação.

7.4. Proposta de (re)definição para o termo arquitetura de computadores

Nesta seção apresenta-se uma proposta de redefinição para o termo "Arquitetura de Computador". Entretanto, em função das dificuldades e da complexidade dessa tarefa, conclui-se que esta redefinição deve ser um processo coletivo (cooperativo e colaborativo) envolvendo inicialmente os representantes das comunidades acadêmica, científica e industrial que atuam no contexto da arquitetura de computadores e posteriormente os representantes das comunidades acadêmica, científica e industrial que atuam no contexto mais amplo da computação.

Este grande desafio, deve ser realizado por um grupo de professores e pesquisadores que trabalharão continuamente e terão encontros nas edições do Workshop sobre Educação em Arquitetura de Computadores WEAC e estarão publicando os resultados iniciais no International Journal of Computer Architecture Education IJCAE.

Deste modo, está seção tem o propósito de iniciar uma discussão sobre a importância, a relevância e a necessidade da redefinição do termo "Arquitetura de Computador".

Neste processo, deve-se inicialmente avaliar e analisar os principais impactos positivos e negativos que uma redefinição poderia produzir, especificamente na educação em arquitetura de computadores e de modo geral e global na educação em computação.

7.4.1. O termo "Arquitetura"

O processo de redefinição do termo "Arquitetura de Computador", é um grande desafio, tanto no contexto da sua própria área do conhecimento, quanto na grande área da computação.

Este processo deve ser iniciado com um amplo estudo e análise da origem e do significado original do termo arquitetura e também dos principais termos relacionados, como: arquiteto, descrição arquitetural, documentação arquitetural, estilo arquitetural, projeto arquitetônico e outros.

Posteriormente, deve-se analisar a evolução e a diversificação do significado do termo arquitetura ao longo do tempo, ao ser utilizado em diversas áreas do conhecimento diferentes da sua área de origem. Neste contexto, uma das utilizações mais destacadas foi justamente na área de computação, considerando-se inicialmente o termo arquitetura de computador.

Posteriormente, deve-se analisar o termo arquitetura na atualidade, considerando as suas diversas utilizações e os seus diferentes significados.

Finalmente deve-se considerar e discutir as principais tendências futuras de utilização e de significado do termo arquitetura.

7.4.2. Proposta de (re)definição para o termo "Arquitetura de Computador"

A recente transição no paradigma de computação predominante, de computação sequencial para computação paralela, produziu uma oportunidade sem precedentes para o início dos processos de revitalização e revalorização da área arquitetura de computadores e da sua educação nos diversos cursos da área de computação.

Entretanto, uma das etapas iniciais e fundamentais destes processos deve ser exatamente discutir e redefinir os principais termos envolvidos, entre os quais, podem ser destacados os termos “computador” e “arquitetura de computador”.

No contexto dessa redefinição, surgem algumas perguntas iniciais que precisam ser analisadas, discutidas e respondidas. Entre as quais, destacam-se:

a) Na prática, para que serve o termo e quem precisa e utiliza o termo?

b) Uma redefinição do termo “arquitetura de computador” é realmente importante e necessária?

c) Quando esta redefinição deve ser realizada?

d) Quem deve realizar esta redefinição?

e) Como esta redefinição deve ser realizada?

No contexto das possíveis respostas, uma análise do estado da arte mostra que os pesquisadores e a própria comunidade científica diretamente relacionada com a arquitetura de computadores não está interessada e não tem se dedicado, em termos quantitativos e qualitativos, à discussão e ao processo de redefinição do próprio termo. Um dos inúmeros exemplos desta situação, é a definição “*Computer Architecture* is the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals. Computer architecture is *not* about using computers to design buildings” apresentada no site “WWW Computer Architecture Page” [18], considerado o principal site relacionado com Arquitetura de Computadores.

Uma análise dos problemas relacionados com os níveis de interesse, motivação e conhecimento dos tópicos relacionados com arquitetura de computadores, mostra que os principais impactos negativos produzidos pelas definições do termo e seus significados ocorrem durante a graduação dos futuros profissionais das áreas da computação.

Deste modo, observa-se que a discussão e uma possível redefinição do termo deverá ser realizada, pelo menos inicialmente, no contexto da comunidade diretamente envolvida e interessada na educação em arquitetura de computadores.

Um aspecto importante é aproveitar a possibilidade de discutir com os próprios criadores do termo [Brooks 1962][Amdahl 1964], que ainda estão vivos, as motivações, as razões e as justificativas da definição original e quais seriam as definições ideais do termo, na atualidade e até no futuro.

O grande desafio é redefinir o termo arquitetura de computador de modo sistêmico, hierárquico e integrador. Considerando-se inicialmente quantos e quais subsistemas (módulos) são necessários para que o computador possa executar os comportamentos desejados de acordo com os principais requisitos funcionais e não funcionais, antes de definir quais subsistemas serão implementados como software ou como hardware. E posteriormente, determinar de modo integrado e hierárquico quais módulos deverão ser implementados como hardware (obrigatório), programável ou não programável, e ou como software (não obrigatório).

Na redefinição do termo “arquitetura de computador” deve-se considerar explicitamente os termos “Arquitetura de Sistema”, “Arquitetura de Hardware” e

“Arquitetura de Software” e os seus significados, dentro de uma abordagem sistêmica e hierárquica.

Posteriormente, deve-se analisar e discutir a necessidade de evolução do significado e da própria definição do termo no futuro, considerando-se as possíveis evoluções dos objetos denominados computadores aos quais o termo arquitetura está relacionado no contexto do termo composto “arquitetura de Computador”.

Finalmente, considera-se que o ideal seria a utilização dos termos “Sistema de Computação” e “Arquitetura de Sistema de Computação” em substituição aos termos “Computador” e “Arquitetura de Computador” e uma nova versão da definição utilizando-se o termo “Arquitetura de Sistema de Computação”. Neste contexto, devem ser considerados os diversos tipos de sistemas de computação incluindo-se os denominados não convencionais ou não tradicionais e algumas das arquiteturas denominadas não convencionais como as arquiteturas reconfiguráveis, arquiteturas evolutivas, arquiteturas inteligentes e arquiteturas autônomas.

Ao finalizar a descrição do grande desafio, algumas perguntas fundamentais devem ser consideradas:

- Uma única definição do termo pode representar de modo ideal toda a diversidade dos objetos denominados “computador”, considerando inclusive as diferentes classes de computadores [9, 12, 13, 16]?

- É possível criar uma definição do termo que posteriormente seja utilizada de modo generalizado, globalizado e até padronizado?

- Quais as vantagens e desvantagens de uma padronização na definição do termo, ou até mesmo, de uma definição padrão do termo “arquitetura de computador”?

Deste modo, pode-se afirmar que a principal meta do grande desafio de redefinição do termo é produzir uma definição mais completa e geral do termo “arquitetura de computadores”, considerando-se as diferentes classes de computadores, independentemente dos seus aspectos de implementação física, e os seus principais aspectos e atributos arquiteturais.

Deve-se considerar que a redefinição do termo será um processo complexo, demorado e até polêmico. Estes aspectos indicam que o processo de redefinição do termo “arquitetura de computadores”, realmente será um grande desafio.

Destaca-se que uma redefinição do termo com possibilidade de aceitação e utilização internacional seria uma contribuição significativa da comunidade brasileira de educação em arquitetura de computadores, inicialmente no contexto de educação em arquitetura de computadores, posteriormente no contexto de arquitetura de computadores e finalmente no contexto geral da computação e da sua educação.

7.5. Atividades de educação (ensino e aprendizado) em arquitetura de computadores

Considerando os impactos negativos das definições do termo Arquitetura de Computadores na educação em Arquitetura de Computadores, apresentados na seção 7.3, apresenta-se nesta seção um conjunto de atividades que foram propostas para melhorar o interesse, a motivação e a qualidade dos conhecimentos aprendidos.

As atividades de ensino e de aprendizado, relacionadas com a educação em arquitetura de computadores, devem ser iniciadas de modo mais informativo e aplicado nos períodos anteriores às disciplinas Arquitetura de Computadores, idealmente no início dos cursos. As atividades devem ocorrer tanto dentro de disciplinas, quanto em atividades interdisciplinares ou multidisciplinares. Posteriormente, atividades com enfoque formativo e também com enfoque aplicado devem ser desenvolvidas nas disciplinas diretamente relacionadas à Arquitetura de Computadores. Finalmente, as atividades com enfoque aplicado, devem ser desenvolvidas nos períodos posteriores às disciplinas diretamente relacionadas à Arquitetura de Computadores.

Apresentam-se vários tipos de atividades, propostas para desenvolvimento nos períodos anteriores às disciplinas Arquitetura de Computadores, nas disciplinas diretamente relacionadas com a arquitetura de computadores e nos períodos posteriores às disciplinas Arquitetura de Computadores. Em cada atividade, existem as subatividades que devem ser executadas pelos professores e as subatividades que devem ser executadas pelos alunos.

No futuro, com a disponibilidade cada vez maior de computadores com diferentes arquiteturas, como arquiteturas reconfiguráveis e outras arquiteturas não convencionais, e até com arquiteturas híbridas, as atividades propostas devem utilizar computadores que possuem os diversos tipos de arquitetura.

7.5.1. Atividades anteriores às disciplinas Arquitetura de Computadores

As atividades desenvolvidas nos períodos anteriores às disciplinas Arquitetura de Computadores devem ocorrer tanto dentro de disciplinas, quanto em atividades interdisciplinares ou multidisciplinares. Na atualidade, deve-se explorar de modo informativo e motivador as diversidades das arquiteturas paralelas dos processadores de propósito geral com múltiplos núcleos. No futuro, deve-se explorar de modo informativo e motivador as diversidades das arquiteturas paralelas dos processadores com múltiplos núcleos, tanto os processadores de propósito geral com múltiplos núcleos, quanto os processadores gráficos ou seja as GPUs e até mesmo os processadores híbridos que integram processadores de propósito geral e processadores gráficos com múltiplos núcleos.

Apresentam-se várias propostas de atividades, e as avaliações e análises dos resultados iniciais produzidos com as primeiras implementações de algumas dessas atividades propostas.

7.5.1.1. Atividades nas disciplinas anteriores

Nas disciplinas iniciais dos cursos, principalmente nas relacionadas com algoritmos e estruturas de dados e com programação, os professores devem mostrar aos alunos os

impactos da arquitetura do computador no desempenho dos programas sendo executados.

Como exemplo desse tipo de atividades, destacam-se:

1- Avaliação de desempenho de programas paralelos, fornecidos pelo professor da disciplina, codificados utilizando-se a linguagem C e a API OpenMP. Os programas devem ser executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas com diversos números de núcleos e de diferentes fabricantes;

2- Modificações nas diretivas OpenMP, sugeridas e orientadas pelo professor da disciplina usando estudos dirigidos, e posterior avaliação de desempenho de programas paralelos codificados utilizando-se a linguagem C e a API OpenMP. Os diferentes programas modificados devem ser executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas com diversos números de núcleos e de diferentes fabricantes;

3- Desenvolvimento de programas paralelos simples, orientados pelo professor da disciplina, e posterior avaliação de desempenho de programas paralelos desenvolvidos utilizando-se a linguagem C e a API OpenMP. Os programas desenvolvidos são executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas com diversos números de núcleos e de diferentes fabricantes.

Em todas as atividades propostas, os alunos devem ser motivados a preparar relatórios e até mesmo artigos nos formatos de artigos de eventos científicos, como por exemplo do WSCAD-WIC e do WSCAD-SSC.

Nos últimos semestres, temos desenvolvidos atividades de avaliação de desempenho de programas paralelos onde os alunos utilizam estudos dirigidos para avaliar e analisar os impactos de otimizações de código, relacionadas com o número de threads utilizados e com a utilização eficiente das memórias cache, no desempenho dos programas paralelo. A maioria dos alunos mostra-se muito interessada e motivada no aprendizado dos detalhes das arquiteturas paralelas dos processadores com múltiplos núcleos, após a verificação na prática e o entendimento dos ganhos de desempenho causados pela utilização dos conhecimentos das arquiteturas paralelas utilizadas na execução dos programas. Os alunos observam e entendem a importância do aprendizado dos tópicos relacionados com a arquitetura de computadores, mesmo e até principalmente para os alunos interessados principalmente ou somente no desenvolvimento de algoritmos e de software. Deste modo, conclui-se que as atividades propostas com enfoque mais informativo, que integram o desenvolvimento de software aplicativo, avaliação de desempenho de software paralelo e arquitetura de computadores motivam os alunos ao início do estudo e do aprendizado dos tópicos relacionados com arquitetura de computadores, mesmo antes das disciplinas diretamente relacionadas com a arquitetura de computadores.

7.5.1.2. Atividades interdisciplinares ou multidisciplinares anteriores

Nos períodos iniciais dos cursos, deve-se envolver tópicos das disciplinas relacionadas com algoritmos e estruturas de dados e das disciplinas relacionadas com matemática e

física. Nestas atividades os professores devem mostrar aos alunos os impactos da arquitetura do computador no desempenho e na qualidade do software sendo executado. Na atualidade, deve-se explorar as diversidades das arquiteturas paralelas dos processadores com múltiplos núcleos.

Como exemplo desse tipo de atividades, destacam-se:

1- Avaliação de desempenho de programas paralelos que realizam cálculos matemáticos simples, como o cálculo de pi ou a manipulação de matrizes, fornecidos pelo professor orientador da atividade, codificados utilizando-se a linguagem C e a API OpenMP. Os programas devem ser executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas com diversos números de núcleos e de diferentes fabricantes. Os alunos devem realizar análises estatísticas dos resultados de desempenho, utilizando medidas estatísticas como diferentes tipos de médias, variância, desvio padrão e coeficiente de variação. Posteriormente, os alunos devem preparar tabelas e gráficos para a apresentação e a análise dos resultados. Finalmente, os alunos devem preparar relatórios e até mesmo artigos nos formatos de artigos de eventos científicos, como por exemplo do WSCAD-WIC.

7.5.2. Atividades nas disciplinas diretamente relacionadas com a Arquitetura de Computadores

Nas disciplinas diretamente relacionadas com a arquitetura de computadores, os professores devem mostrar aos alunos que os conhecimentos relacionados com a arquitetura dos computadores podem produzir impactos significativos no desempenho dos algoritmos e dos programas executados nos computadores, com diferentes arquiteturas, utilizados nas atividades. Além disso, os professores devem mostrar aos alunos que os conhecimentos dos diferentes tópicos relacionados com a arquitetura dos computadores também podem produzir ganhos significativos na qualidade dos resultados produzidos pelos programas (algoritmos e estruturas de dados) executados nos diferentes computadores utilizados nas atividades.

Com a realização das atividades propostas para os períodos anteriores, apresentadas em 7.5.1, muitos alunos podem se motivar e iniciar um aprendizado antecipado durante os períodos anteriores. Entretanto, esta motivação pode diminuir, durante as disciplinas diretamente relacionadas à arquitetura de computadores, se não bem executadas, a ponto de prejudicar o interesse e até mesmo a qualidade dos conhecimentos aprendidos durante essas disciplinas.

Na atualidade, nestas atividades, deve-se explorar com maior profundidade as diversidades das arquiteturas paralelas dos processadores com múltiplos núcleos e também as diversidades das arquiteturas paralelas dos sistemas de computação, envolvendo de modo isolado e integrado os elementos de processamento, os elementos de armazenamento e os elementos de comunicação. No futuro, deve-se explorar de modo formativo as diversidades das arquiteturas paralelas dos processadores com múltiplos núcleos, tanto os processadores de propósito geral, quanto os processadores gráficos com múltiplos núcleos e também os processadores híbridos que integram processadores de propósito geral e processadores gráficos com múltiplos núcleos.

Como exemplo desse tipo de atividades, destacam-se:

1- Avaliação de desempenho de programas seqüenciais, fornecidos pelo professor da disciplina, codificados utilizando-se a linguagem C com o propósito de explorar os aspectos arquiteturais dos computadores, como memória cache e pipeline. Os programas devem ser executados em computadores que utilizam diferentes modelos de processadores, que possuem diferenças em relação à hierarquia de memória cache e outros aspectos arquiteturais e de diferentes fabricantes;

2- Desenvolvimento de programas seqüenciais, orientados pelo professor da disciplina, e posterior avaliação de desempenho de programas desenvolvidos utilizando-se a linguagem C com o propósito de explorar aspectos arquiteturais dos computadores como memória cache e pipeline. Os programas desenvolvidos devem ser executados em computadores que utilizam diferentes modelos de processadores, que possuem diferenças em relação à hierarquia de memória cache e outros aspectos arquiteturais e de diferentes fabricantes;

3- Avaliação de desempenho de programas paralelos, fornecidos pelo professor da disciplina, codificados utilizando-se a linguagem C e a API OpenMP o propósito de explorar os aspectos arquiteturais dos computadores, como número de núcleos, memória cache e pipeline. Os programas devem ser executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, com diversos números de núcleos, e que possuem outras diferenças arquiteturais em relação à hierarquia de memória e outros aspectos arquiteturais e de diferentes fabricantes;

4- Modificações nas diretivas OpenMP, sugeridas e orientadas pelo professor da disciplina usando estudos dirigidos, e posterior avaliação de desempenho de programas paralelos codificados utilizando-se a linguagem C e a API OpenMP. Os diferentes programas modificados devem ser executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas e de diferentes fabricantes;

5- Desenvolvimento de programas paralelos simples, orientados pelo professor da disciplina, e posterior avaliação de desempenho de programas paralelos desenvolvidos utilizando-se a linguagem C e a API OpenMP. Os programas desenvolvidos são executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas e de diferentes fabricantes.

Em todas as atividades propostas, os alunos devem ser motivados a preparar relatórios nos formatos de artigos de eventos científicos, como por exemplo do WSCAD-WIC e do WSCAD-SSC. Os professores devem avaliar todos os artigos preparados pelos alunos e motivar os autores dos melhores artigos à submeter os mesmos para o WSCAD-WIC.

Nos últimos semestres, temos desenvolvido algumas dessas atividades na disciplina Arquitetura de Computadores III, que é uma disciplina do quarto período do curso Ciência da Computação da PUC Minas. Atualmente, todos os alunos preparam relatórios no formato de artigo do WSCAD-WIC, que são avaliados pelo professor da disciplina. Dentro os melhores artigos preparados e submetidos pelos alunos, muitos têm sido aprovados e publicados nos últimos anos no WSCAD-WIC.

A grande maioria dos alunos mostra-se muito interessada, motivada e convencida

que o domínio dos conhecimentos relacionados com a arquitetura de computadores, principalmente das arquiteturas paralelas dos processadores com múltiplos núcleos, produz na prática, ganhos de desempenho significativos na execução dos programas. Os alunos comprovam de modo formativo a importância do aprendizado dos tópicos relacionados com a arquitetura de computadores. Estes resultados observados, independem dos interesses futuros dos alunos, inclusive dos alunos interessados no desenvolvimento de algoritmos, desenvolvimento de software e até em engenharia de software. Sendo que os últimos, demonstram interesse inicial em engenharia de software paralelo. Deste modo, conclui-se que as atividades com enfoque formativo que integram o desenvolvimento de software aplicativo, avaliação de desempenho de software seqüencial e paralelo e arquitetura de computadores motivam os alunos à continuidade e ao aprofundamento do estudo e aprendizado dos tópicos relacionados com arquitetura de computadores.

7.5.3. Atividades posteriores às disciplinas Arquitetura de Computadores

As atividades desenvolvidas nos períodos posteriores às disciplinas Arquitetura de Computadores devem ocorrer tanto dentro de disciplinas, quanto em atividades interdisciplinares ou multidisciplinares. Apresentam-se várias propostas de atividades, e as avaliações e análises dos resultados iniciais produzidos com as primeiras implementações de algumas dessas atividades propostas.

7.5.3.1. Atividades nas disciplinas posteriores

Nas disciplinas finais dos cursos, principalmente nas que são relacionadas à engenharia de software e às principais áreas de aplicação, os professores devem mostrar aos alunos os impactos da arquitetura do computador no desempenho dos programas de cada área de aplicação sendo executados. Na atualidade, deve-se explorar as diversidades das arquiteturas paralelas dos processadores com múltiplos núcleos, tanto os processadores de propósito geral com múltiplos núcleos quanto, se possível, os processadores gráficos ou seja as GPUs.

Como exemplo desse tipo de atividades, destacam-se:

1- Avaliação de desempenho de programas paralelos relacionados com a disciplina, fornecidos pelo professor da disciplina, codificados utilizando-se a linguagem C e a API OpenMP. Os programas devem ser executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas com diversos números de núcleos e de diferentes fabricantes;

2- Modificações nas diretivas OpenMP, sugeridas e orientadas pelos professores da disciplina e da disciplina arquitetura de computadores, usando estudos dirigidos, e posterior avaliação de desempenho de programas paralelos codificados utilizando-se a linguagem C e a API OpenMP. Os diferentes programas modificados devem ser executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas com diversos números de núcleos e de diferentes fabricantes;

3- Desenvolvimento de programas paralelos simples e relacionados com a disciplina, orientados pelos professores da disciplina e da disciplina arquitetura de computadores, e posterior avaliação de desempenho de programas paralelos desenvolvidos utilizando-se

a linguagem C e a API OpenMP. Os programas desenvolvidos são executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas com diversos números de núcleos e de diferentes fabricantes.

Em todas as atividades propostas, os alunos devem ser motivados a preparar relatórios nos formatos de artigos de eventos científicos, como por exemplo do WSCAD-WIC, WSCAD-SSC e nos eventos científicos diretamente relacionados com as áreas de interesse de cada aluno.

Nos últimos semestres, iniciamos o desenvolvimento de atividades de avaliação de desempenho de programas paralelos relacionados com áreas de aplicação como processamento digital de imagens e inteligência computacional. onde os alunos desenvolvem programas paralelos utilizando diferentes linguagens e APIs para a avaliação e a análise dos impactos de otimizações de código, relacionadas com o número de *threads* utilizados e com a utilização eficiente das memórias cache, no desempenho dos programas paralelo. Os alunos envolvidos nas atividades mostraram-se interessados e motivados no aprofundamento do aprendizado de arquiteturas paralelas dos processadores com múltiplos núcleos, para o aumento dos ganhos de desempenho causados pela utilização dos conhecimentos das arquiteturas paralelas utilizadas na execução dos programas. Os alunos observam e entendem a importância do aprendizado dos tópicos relacionados com a arquitetura de computadores, mesmo e até principalmente para os alunos interessados principalmente ou somente no desenvolvimento de algoritmos e de software. Deste modo, conclui-se que as atividades propostas com enfoque mais informativo, que integram o desenvolvimento de software aplicativo, avaliação de desempenho de software paralelo e arquitetura de computadores motivam os alunos à continuidade e aprofundamento do estudo e do aprendizado dos tópicos relacionados com arquitetura de computadores, mesmo após as disciplinas diretamente relacionadas com a arquitetura de computadores.

Os alunos das disciplinas compiladores e sistemas operacionais deveriam utilizar os conhecimentos relacionados com os tópicos de AC nas atividades desenvolvidas nas respectivas disciplinas. Muitas vezes, os alunos de compiladores não trabalham com os tópicos relacionados com a geração de código para uma arquitetura específica ou idealmente para mais de uma arquitetura específica.

7.5.3.2. Atividades interdisciplinares ou multidisciplinares posteriores

Nas disciplinas finais dos cursos, principalmente nas relacionadas com engenharia de software e com as principais áreas de aplicação, os professores devem mostrar aos alunos os impactos da arquitetura do computador no desempenho dos programas de cada área de aplicação sendo executados. Na atualidade, deve-se explorar as diversidades das arquiteturas paralelas dos processadores com múltiplos núcleos, tanto os processadores de propósito geral com múltiplos núcleos quanto, se possível, os processadores gráficos ou seja as GPUs.

Como exemplo desse tipo de atividades, destacam-se:

1- Avaliação de desempenho de programas paralelos que realizam cálculos matemáticos simples, como o cálculo de pi ou a manipulação de matrizes, fornecidos pelo professor orientador da atividade, codificados utilizando-se a linguagem C e a API OpenMP. Os

programas devem ser executados em computadores que utilizam diferentes modelos de processadores com múltiplos núcleos, que possuem diferentes arquiteturas paralelas com diversos números de núcleos e de diferentes fabricantes. Os alunos devem realizar análises estatísticas dos resultados de desempenho, utilizando medidas estatísticas como diferentes tipos de médias, variância, desvio padrão e coeficiente de variação. Posteriormente, os alunos devem preparar tabelas e gráficos para a apresentação e a análise dos resultados. Finalmente, os alunos devem preparar relatórios e até mesmo artigos nos formatos de artigos de eventos científicos, como por exemplo do WSCAD-WIC.

Nestas atividades, deve-se explorar com maior profundidade as diversidades das arquiteturas paralelas dos processadores com múltiplos núcleos e também as diversidades das arquiteturas paralelas dos sistemas de computação, envolvendo de modo isolado e integrado os elementos de processamento, os elementos de armazenamento e os elementos de comunicação.

Aprendizado dos tópicos de modo integrado com atividades de projeto e desenvolvimento de algoritmos e softwares com diferentes requisitos funcionais e requisitos não funcionais para execução em diferentes classes de sistemas de computação. Neste contexto, os alunos são motivados a tomadas de decisão relacionadas com os modelos de computação e com as arquiteturas de sistemas de computação mais indicadas para a solução computacional de cada problema.

7.6. Conclusões

De modo global, espera-se que este capítulo seja utilizado como texto de referência sobre a origem, evolução e atualidade do termo “Arquitetura de Computadores” e como texto de suporte para análises e reflexões sobre a educação em Arquitetura de Computadores e indiretamente sobre a própria arquitetura de computadores.

Apresentou-se neste capítulo, uma análise da origem e da evolução ao longo dos anos das principais definições do termo “Arquitetura de Computador”, que foram apresentadas na literatura. Pretende-se futuramente, a preparação de uma versão mais elaborada dessa análise, no formato de um artigo, que será submetido para um periódico da área de arquitetura de computadores

Posteriormente, apresentou-se uma proposta de (re)definição para o termo “Arquitetura de Computadores” que considera de modo integrado e hierárquico as arquiteturas do sistema de computação, do software e do hardware. Considera-se que este processo de redefinição é um grande desafio, e espera-se que o conteúdo dessa seção seja um motivador e material de referência inicial desse grande desafio.

Posteriormente, apresentou-se com conjunto de sugestões de atividades relacionadas ao ensino e aprendizado de arquitetura de computadores que utilizam as definições e significados do termo de modo motivador. Os resultados iniciais, obtidos com a utilização de algumas propostas de atividades apresentadas, já produziram ganhos significativos em termos de motivação, interesse e qualidade do aprendizado de arquitetura de computadores.

Finalmente, espera-se que esse capítulo contribua, cada vez mais, nos aumentos de interesse, motivação e qualidade no aprendizado da arquitetura de computadores.

References

Amdahl, G. M.; Blaauw, G. A.; Brooks, F. P. Jr. (1964) “Architecture of the IBM System/360” IBM Journal of Research and Development, vol. 8, no. 2, april 1964, p.87-101.

Anselmo, D. N.; Mendes, I. T.; Silveira, L. G. H. D.; Carvalho, M. B. Campos Jr, R. O.; Martins, C. A. P. S. ”Proposta de Criação de uma Revista Eletrônica Sobre Educação em Arquitetura de Computadores”, Workshop sobre Educação em Arquitetura de Computadores - WEAC, 2010, p. 13-17

Barton, R. S. (1961) "Functional Design of Computers", Communications of the ACM, v. 4, n. 9, p. 405- ,1961

Barton, R. S. (1961) "A New Approach to the Functional Design of a Digital Computer", Proceedings of the Western Joint Computer Conference, May 1961, p.393-396.

Bell, C. G.; Newell, A. (1971) “Computer Structures: Readings and Examples”, McGraw-Hill

Brainerd, J. G., and Sharpless, T. K. "The ENIAC." *Electrical Engineering,* 67 (2) p. 163-172, (1948), Reprinted 1999 in Proceedings of the IEEE, 87(6), 1031–1043.

Brooks, F. P. Jr. (1962) “Architectural Philosophy” in *Planning a Computer System,* W. Buchholz, ed., McGraw-Hill, 1962, p.5-16.

Buchholz, W. (1962) “Planning a Computer System: Project Stretch”, McGraw-Hill, 1962

Burks, Arthur W. (Walter), “Electronic Computing Circuits of the ENIAC”, Proceedings of the Institute of Radio Engineers (I.R.E.) August 1947, vol 35, n 8, pp. 756-767

Burks, Arthur W; Burks, Alice R. (1981) The ENIAC: The First General-Purpose Electronic Computer, IEEE Annals of the History of Computing, vol 3, n. 4, p. 310-399, 1981

Ching, F. D. K. (1996) “A Visual Dictionary of Architecture”, Wiley, 1996

Ching, F. D. K.; Jarzombek, M. M.; Prakash, V. (2006) “A Global History of Architecture”, Wiley, 2006

Eckert, Jr., J. Presper; Weiner, James R.; Welsh, H. Frazer; Mitchell, Herbert F. (1951), The UNIVAC System, *International Workshop on* Managing Requirements Knowledge (AFIPS), 1951, p. 6-16

Fazio, M.; Moffett, M.; Wodehouse, L. (2008) “A World History of Architecture”, 2 edition, McGraw-Hill, 2008

Fletcher, B. (1956) “A History of Architecture on the Comparative Method”, Charles Scribner's Sons, 1956

Godfrey, M.D.; Hendry, D.F. (1993) The Computer as von Neumann Planned It IEEE Annals of the History of Computing, Vol. 15, No. 1, 1993, pp. 11-21

Goldstine, A. *A Report on the ENIAC (Electronic Numerical Integrator and Computer), Part 1. Technical Description of the ENIAC, Ballistics Research Laboratory, 1946*

Goldstine, H. H.; Goldstine, A. "The Electronic Numerical Integrator and Computer (ENIAC)", *Mathematical Tables and Other Aids to Computation* Vol. 2, No. 15 (Jul., 1946), pp. 97-110 - Reimpressão Goldstine, H. H.; Goldstine, A. "The Electronic Numerical Integrator and Computer (ENIAC)", Annals of the History of Computing, IEEE, Vol 18, n 1, pp. 10-16

Hennessy, J. L.; Patterson, D. A. (2006) "Computer Architecture: A Quantitative Approach", 4 edition, Morgan Kaufmann, 2006

Hennessy, J. L.; Patterson, D. A. (2012) "Computer Architecture: A Quantitative Approach", 5 edition, Morgan Kaufmann, 2012

Hill, M. D.; Jouppi,N.P.; Sohi, G. S. (2000) "Readings in Computer Architecture", Morgan Kaufmann, 2000

Martins, C. A. P. S. "Redefinição do Termo Arquitetura de Computador", Workshop sobre Educação em Arquitetura de Computadores - WEAC, 2009, p. 92-95.

Martins, C. A. P. S. "Impactos na área da computação causados pela heterogeneidade de modelos de arquitetura, materiais e princípios físicos utilizados na implementação de sistemas de computação", II seminário sobre Grandes Desafios da Computação no Brasil, 2009, p. 1-3.

Muller, D. E. "The place of logical design and switching theory in the computer curriculum", Communications of the ACM, v. 7, n. 4, April 1964, ACM, p.222 - 225

Null, L. (2006) "The Essentials of Computer Organization and Architecture", 2 edition, Jones & Bartlett Pub 2006

Patterson, D. A.; Hennessy, J. L. (2008) "Computer Organization and Design: The Hardware/Software Interface", 4 edition, Morgan Kaufmann, 2008

Pugh, E. W.; Johnson, L. R.; Palmer, J. H. (1991) "IBM's 360 and Early 370 Systems", MIT Press, 1991

Smith, R. E. "A Historical Overview of Computer Architecture" IEEE Annals of the History of Computing, v. 10, n. 4, oct. 1988, p.277-303

Stallings, W. (2005) "Computer Organization and Architecture: Designing for Performance", 7 edition, Prentice Hall, 2005

Stern, Nancy (1981), "From ENIAC to UNIVAC, An appraisal of the Eckert-Mauchly Computers", Bedford, Massachusetts: Digital Press

Silva, A. R.; Valadão, E. F. F.; Figueiredo, R. S.; Martins, C. A. P. S. "Abordagem prática fundamentada em análises e modificações de programas paralelos para motivação ao aprendizado de Arquitetura de Computadores", Workshop sobre Educação em Arquitetura de Computadores - WEAC, 2010, p. 9-12

Von Neumann, J. (1945), "First Draft of a Report on the EDVAC" Moore School of Electrical Engineering, University of Pennsylvania, June 30, 1945. Corrected and

complete version published in IEEE Annals of the History of Computing, vol. 15, No.4, 1993, pp. 27-75.

Wiki (2012) “Architecture” http://en.wikipedia.org/wiki/Architecture

Wilkes, M. V. (1951), The EDSAC Computer, International Workshop on Managing Requirements Knowledge (AFIPS), 1951

Wilkes, M.V. (1975) "Early computer developments at Cambridge: The EDSAC", Radio and Electronic Engineer, vol 45, n. 7, pp. 332-335

Williams, M. R. (1993), “The Origins, Uses, and Fate of the EDVAC”, IEEE Annals of the History of Computing. Vol. 15. No. 1. Jan., pp. 22-38.

WWW Computer Architecture Page (2012) http://www.cs.wisc.edu/arch/www/