Pôle Universitaire Français à HoChiMinh ville -

Laboratoire de Recherche et Développement de l'EPITA

Finalizing a benchmarking system for Common Lisp

Nguyen Duc Tung

November, 2010

Abstract

This project is in the context of an ongoing research on the behavior and performance of Common Lisp. The current system produces large datasets of benchmarks. The purpose of this project is to improve the current system in order to make it robust and extensible for future use. The new system uses raw text files as databases for representing data graphically. The GUI is written in Common Lisp and built on top of McCLIM. The exact database format is described in a configuration file that can be edited, which makes the system extensible. The application loads data and filters it out on a GUI for visualization. Diagrams or comparison charts can be also exported according to users' selections on the GUI by using gnuplot.

Contents

1 Introduction

Nowadays, programmers have a wide choice of programming languages. One of the criteria for choosing the programming language is the performance. Common Lisp[1] is not a prime candidate because of misinformation regarding its level of performance. To take an example from practice, in image processing domain, libraries are predominantly written in C (Forment, 2000)[1] or in C++ with various degrees of genericity (Duret-Lutz, 2000)[2].

To show people that they would lose nothing in performance by using Common Lisp, the corollary being that they would gain a lot in expressiveness, Didier Verna is conducting an experimental research on the behavior and performance of Lisp. The study consists in three main parts: arithmetic operation (Verna, 2006)[3], dynamic object-orientation (Verna, 2009)[4], meta-programming. The first part was published, and the second one is in the process.

The benchmarking systems produce a vast number of test cases (micro-benchmarks) because several compilers are being tested with different parameters involved (data type, initialization, class size, optimization level, etc.). The management of the results needs to be robust as well as easy for future use. Moreover, in the current state of the benchmarking system, Xmgrace[2] is used for generating the comparison chart. Xmgrace is not a stable tool, and not unified. Its interface has changed from version to version. Xmgrace5 and Xmgrace6 are very different. Even though in the same version, for instance, Xmgrace5, the build Xmgrace5.99.1 is not compatible with the build Xmgrace5.99.0. So, Xmgrace is not an ideal tool for a potent system.

The purpose of my work, in co-operation with Ha Minh Vuong, is to help Didier improve his benchmarking system by proposing a database format to store/retrieve the benchmark results, and generate raw text files that are used by gnuplot[3] to create comparison diagrams. Gnuplot is more mature than Xmgrace as to our point of view[4].

I and Ha Minh Vuong decide to use raw text files for the database rather than a relational database management system because of the following advantages: it is light, easy to manipulate, does not require any additional library. Since we have enormous datasets, when data is loaded on the GUI, it is filtered out according to parameters that are chosen for visualization and generation of the comparison chart using gnuplot. The application can export the raw text file based on parameter selection. This allows users to precisely customize what appears on the graphs.

[1] Common Lisp is defined by the X3.226-1994 (R1999) ANSI standard.

[2] ACE/gr is a 2D plotting tool for the X Window System. It uses an OSF Motif-based user interface, which is the reason why it's also known as Xmgr. http://plasma-gate.weizmann.ac.il/Xmgr

[3] Gnuplot is a portable command-line driven graphing utility for linux, OS/2, MS Windows, OSX, VMS, and many other platforms. http://www.gnuplot.info

[4] Section gnuplot. Finalizing a benchmarking system for Common Lisp, Ha Minh Vuong.

1.1 On the Behavior and Performance of Lisp, Part 1

This is the first part in a series of experimental studies on the behavior and performance of Common Lisp by Didier Verna[5].

The article experiments are based on four very simple algorithms: pixel assignment (initializing an image with a constant value), and pixel addition/ multiplication/ division (filling a destination image with the addition/ multiplication/ division of every pixel from a source image by a constant value). These algorithms involve two fundamental atomic operations of image processing: pixel (array element) access and arithmetic processing. Each algorithm runs on 200 consecutive iterations to avoid benchmarking any program initialization side effect (initial page faults etc.). The following parameters are used:

- Image size: 2 sizes of image are used with different iterations. The normal image size is 800 * 800 runs with 200 iterations, and the small image is 400 * 400 runs with 800 steps.
- Data type: both integer and floating-point images are benchmarked with corresponding arithmetic operations.
- Array types: 2D images represented either directly as 2D arrays, or as 1D arrays of consecutive lines.
- Access type: both linear and pseudo-random image traversal are benchmarked. In linear image traversal, the images are traversed from the first pixel to the last, line after line, as they are represented in memory. In pseudo-random image traversal, the images are traversed by examining in turn pixels distant from a constant offset in the image memory representation.
- Optimization: we have 3 different optimization levels: unoptimized, optimized, and also with inlining of each algorithm's function into the iterations loop.

1.2 On the Behavior and Performance of Lisp, Part 2.1 CLOS Efficiency: Instantiation

The second part deals with CLOS, the Common Lisp Object System. The purpose is to evaluate the behavior and efficiency of instantiation, slot access and generic dispatch in general. This paper (Verna, 2009)[4] describes the results of the experiments on instantiation. A number of parameters are needed to test the behavior and performance of instantiation in as many configurations as possible. We go through to these parameters:

- Class size: For each class size N, the actual class(es) are constructed in 3 different ways, defined as follows.

[5]Didier Verna - http://www.lrde.epita.fr/~didier

  - Plain: A single class containing all the N slots.
  - Vertical: A hierarchy of N + 1 classes containing one slot each, and inheriting from one upper class at a time, the toplevel class being a class with no slot.
  - Horizontal: A set of N simple classes containing one slot each, and a final class with no slot, inheriting (via multiple inheritance) directly from these N upper classes.

- Slot type: we test both int/ floats slot in the C++ case, and fixnum/ single-float one in Common Lisp case.
- Slot Allocation: local (instance-wide) and shared (class-wide) slots are used. Slot Initialization: we have the slots are initialized or leave them uninitialized.
- Optimization Level: "Safe", "Optimized" and "Inlined" optimization modes are also used for benchmark.

2 EPITA and LRDE

2.1 EPITA

The *École Pour l'Informatique et les Techniques Avancées* (the Graduate School of Computer Science and Advanced Technologies), mostly known as EPITA, is a French *Grande École*. Beside university, the "*Grande École*" system constitutes an alternate cursus for higher education in France. The school was established in 1984, located in Kremlin Bicetre - south of Paris.

EPITA is specialized in the fields of Computer Science and software engineering. The students first have two years of preparatory class. They study mathematics, physics, electronics, algorithms and computer science. After this period, students learn the fundamentals of Information Technology and Software Engineering. Then they choose one of the seven majors of the school (System – Network – Security, Telecommunication, Information System – Software Engineering, etc.).

EPITA has a system of laboratories for research. One of them is the *Research and Development Lab, LRDE*[6] *– Laboratoire de Recherche et Développement de l'EPITA*. This is the laboratory where the author did the internship.

2.2 LRDE

LRDE was established in the beginning of 1998 at EPITA. This is the lab where best students of EPITA would join in to contribute their research; ENST[7],

[6] http://www.lrde.epita.fr
[7] Telecom Paris Tech - http://www.telecom-paristech.fr

CNRS/ INRIA[8], and Institut Curie collaborate with the lab in many fields: finite state machine manipulation platform with ENST, Robotics with CNRS/ INRIA, and with Institut Curie, LRDE cooperates to do research about recognition of cancerous cells. Beside, LRDE also has a lot of researching with industry, like pattern recognition with the French Atomic Energy Agency CEA, distributed system within a multi-agent framework dedicated to mobile services with Bouygues Telecom and etc.

LRDE covers various fields of computer science:

- Image and signal processing.
- Automata theory.
- Metaprogramming.
- Software engineering.
- Distributed software, etc.

Below are projects which all members of the laboratory concentrate to develop, improve and contribute to the science community:

- Olena – Image processing, C++ static programming.
- Vaucanson – Finite state machine, C++ generic programming.
- Transformers – Parsing, Program transformation, C++ / Metaprogramming, Optimization.
- Tiger – A Tiger compiler in C++.
- Spot – A model checking library, LRDE is in charge of maintaining with MoVe team - LIP6[9].

Furthurmore, many members of the laboratory are involved in the free software community. They are maintainers of GNU Autoconf, GNU Automake, GNU Bison, XEmacs etc.

3 Requirements

As can be seen after running the benchmarks, the system generates a lot of datafiles. Thus, finding the desired results for a particular benchmark is difficult. The purposes of my work, in collaboration with Ha Minh Vuong, are to make the benchmarking system more robust and extensible as well as making it easier to retrieve and view the results.

[8]INRIA The French National Institute for Research in Computer Science and Control - http://www.inria.fr

[9]LIP6 MoVe - http://move.lip6.fr

The following requirements should be met:

- The new system must be useable not only for the two current sets, but also for other ones later.
- The new system must make it easy to browse the results according to all available parameters.

For this situation, using a database is a good solution. Two options were considered:

- Using a database management system.
- Using an ad-hoc one.

After discussing and analyzing the advantages and disadvantages of these two posibilities, we decided to use a structured text file for the following reasons:

- It is more robust. We do not need to set up a database management system on the machine to connect the database system to our application. We just need our application alone.
- It is more portable. We can take the database everywhere; and use a text editor to view the content, although it does not look nice.

The only difficulty for this choice is to build a new tool so that it can load and retrieve data from the database file. And the idea is to build a Graphical User Interface (GUI) application.

4 Choosing the language

The author chose Common Lisp for several reasons:

- Common Lisp is a standardized language. This means that several implementations are available, including open source and free ones; and there is no risk that the language is either changed in radical ways or is not continued to develope.
- It has its own community from which it is easy to get support.
- The features of Common Lisp allow the author to implement the requirements of section 3 easily.
- Most of the time, Common Lisp code is shorter and more concise than in other languages. For example, to define classes and accessors, we just need a couple of lines of code:

```
(defclass foo ()
   ((%name :initarg :name :reader name)
    (%type :initarg :type :reader type)))
```

5 Environments

There are a lot of implementations[10] of Common Lisp. The author use SBCL[11], Emacs[12] for edition, and SLIME[13] for interaction.

In Common Lisp, McCLIM[14] is the best choice to implement the GUI. McCLIM is an open source library, which implements the specification of the Common Lisp Interface Manager - CLIM[15]. Though, in order for McCLIM to run properly, we need more libraries installed:

- Flexichain [16]
- Spatial-trees [17]
- CLX [18]

6 Techniques and analysis

6.1 The structure of the configuration file

One of the requirements of the new system is the extensibility for future use. This means that the GUI application must have the ability to understand the parameters of the future benchmarks and load them correctly. Since each benchmarking system has several parameters to combine into algorithms, in order to satisfy this requirement, the system needs a file to describe these values, it is called the configuration file. Hence, each database needs a corresponding configuration file for it.

When the GUI application loads a database, it also needs to load the configuration file for that database first.

The configuration file defines all the combinations to create benchmarking algorithms. The GUI application reads all these values and displays them.

Moreover, in the database there are a lot of "mysterious" characters[19], such as "i", "f", "o", and so on. They are short names which were chosen by the users to stand for "Integer", "Float", "Optimized", etc. On the other hand, this file also lets the application know the short name of a combination in order for the

[10] Comparison of actively developed Common Lisp implementations - http://www.cliki.net/Common%20Lisp%20implementation

[11] Steel Bank Common Lisp (SBCL) is a high performance Common Lisp compiler. It is open source/free software, with a permissive license - http://www.sbcl.org

[12] GNU Emacs is an extensible, customizable text edito, and more - http://www.gnu.org/software/emacs

[13] SLIME is a Emacs mode for Common Lisp development - http://common-lisp.net/project/slime

[14] A GUI toolkit for Common Lisp - http://common-lisp.net/project/mcclim

[15] CLIM - http://www.kantz.com/jason/clim-primer

[16] An API for editable sequences - http://common-lisp.net/project/flexichain/

[17] A library for accesing dynamic index data structures - http://www.cliki.net/spatial-trees

[18] Common Lisp X window system - http://www.cliki.net/CLX

[19] A sample record from the '09 system: sbcl,sbcl 1.0.35,Linux 2.6.32-24-generic i686,25000000,3487424498,NIL,f,i,s,NIL,std,i,NIL,1.265,2.581,12.325

comparisons of filters to work correctly. And all full names are also displayed well on the GUI application.

This is the content of the configuration file of the '09 system:

```
("Initialization" "Initialized:i" "Uninitialized:u")
("Type" "Untyped:u" "Integer:i" "Float:f")
("Allocation" "Instance:i" "Class:c")
("Hierarchy" "Single:s" "Horizontal:h" "Vertical:v")
("Struct" "Struct:struct" "Substruct:substruct")
("Class" "STD:std" "Meta:meta")
("Level" "Unoptimized:u" "Optimized:o" "Inlined:i")
```

The configuration has seven lines which correspond to all available combinations for benchmarking algorithms. Each line begins with an open parenthesis and ends with a closed parenthesis. All of values are in quotation marks (“”) and separated with spaces.

The first items in the lists are kinds of combination. The remaining items are their values. For example, “Allocation” has two values, “Instance” and “Class”. The colons (:) are used to make the definitions of short names. For instance, the value “Intstance:i” shows that “Instance” is the true value and is displayed on the GUI application, while the letter “i” presents its short name in its slot in the database. It is the same for “Class:c” and so on. If the users would like to organize the short names are as the same as the full name, they can do as normal way by "Name:Name", or users only need to define them as follow "Name". By default, the application understands that if a definition does not have any colons, the short names are the same as their full names.

6.2 The Graphical User Interface application

6.2.1 The CLIM features

The Common Lisp Interface Manager, CLIM, is a Lisp-based programming interface for constructing the graphical user interface of a Lisp application. Therefore, it inherits a lot of the features from Common Lisp. This helps the size of code is reduced, the speed of implementing is increased and the flow of thought is easy to carry out, etc.

CLIM is designed to operate independently from implementations of Common Lisp and window system. This feature makes the application have the portability. The application is more robust, the users can take it and run it on any machines with any host systems.

CLIM provides many of facilities, completely components for the programmers. It provides a lot of buiding options to help the programmers either to build up an user interface quickly or to define their own programming to make an interface suitable for their needs.

6.2.2 The presentation types

There is a data type in CLIM II specification[20] that the author uses to implement this application, called presentation type[5].

In CLIM, a presentation is an area on the screen which is associated with a type and objects of that type. When a command is invoked with particular arguments, all objects of that type are activated and we can select them. When we define a class, that means we define a presentation and a type. Therefore, an object will be retrieved after selecting its representation with the pointer on the display. There is another way to define a presentation type, that is to use the function `define-presentation-type`.

This facility provides a very powerful user interface.

Besides this, the author needs to use other components of CLIM to deliver a complete application. These are menu bar, pane, etc. All of the specifications of these components are described in the CLIM II specification.

6.2.3 An example of the presentation types

Here is a short example of the presentation type. In this example, we define two classes, one called "lang-libs" and the other is "essence". These classes have only one slot which is used to name the objects.

```
(defclass lang-libs ()
   ((%name :initarg :name :reader name)))
(defclass essence ()
   ((%name :initarg :name :reader name)))
```

We create two "lang-libs" instances, named "Lisp" and "CLIM"; and two "essence" instances, named "is fast" and "is robust".

Then we make a function, named "assign", which needs 2 arguments when being invoked. It needs the input of "lang-libs" and then "essence".

```
(define-assign-command (assign :name t)
   ((lang-libs 'lang-libs :prompt "enter a lang-libs")
    (essence 'essence :prompt "enter an essence"))
   (format (find-pane-named *application-frame* 'int) "~a ~a~%"
           (name lang-libs) (name essence)))
```

First, all objects, here "Lisp" and "CLIM", will be activated when the function is called. The program waits for us to click one of those who belong to class "lang-libs". After that, "is robust" and "is fast" are clickable. Finally, the program will print the sentence "Lisp is fast", if we chose "Lisp" and "is fast".

[20] CLIM Spec - http://bauhh.dyndns.org:8000/clim-spec/index.html

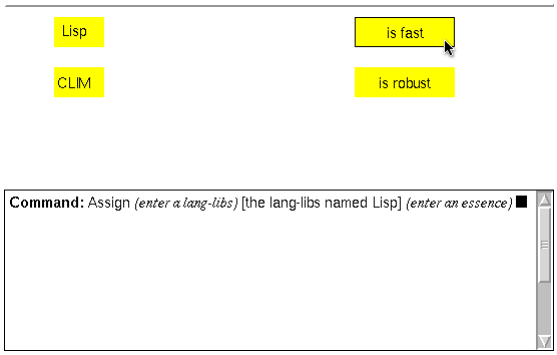

Figure 1: Presentation types example

6.2.4 The design

The GUI application operates as shown below:

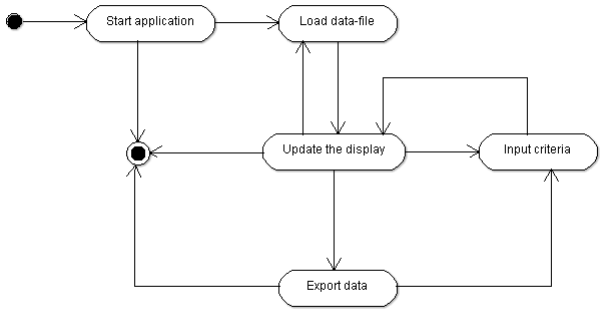

Figure 2: System workflow

The application has 5 areas, which are:

- Compiler names – and versions: displays all of compilers and their versions.
- Architectures: lists all architectures on which the benchmarks were run.
- Algorithm-combinations: lists all available combinations for the database.
- Users can select all those parameters and see the available benchmarks, which are displayed in the Available-results area.
- The remaining part is a box that echoes user-activity, and it is also a place where users can input the path to load the database or to save the filtered data.

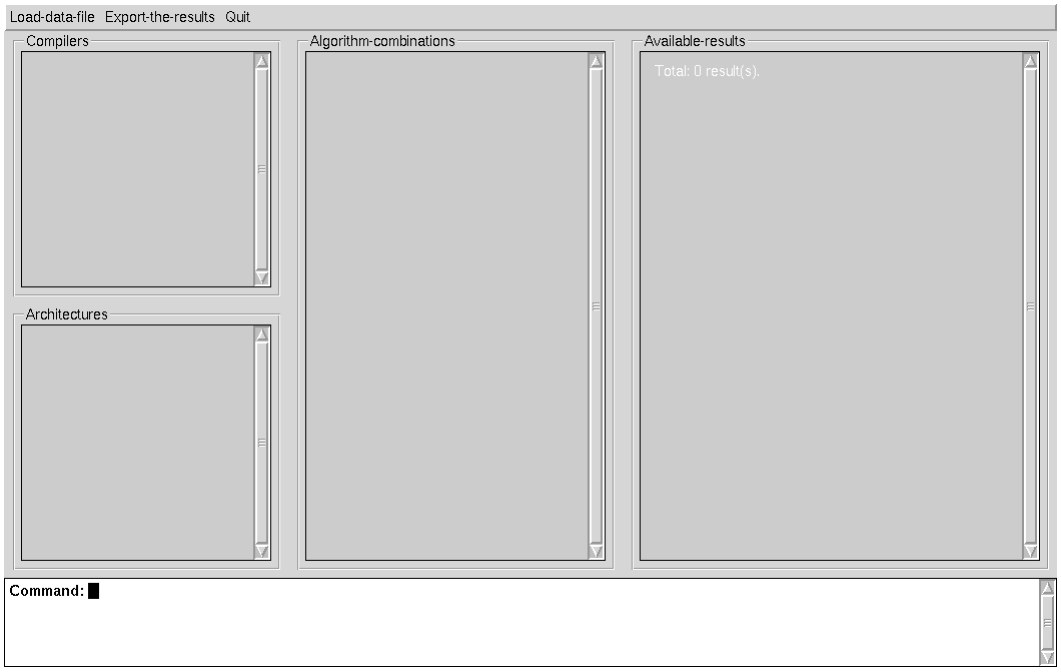

Figure 3: Benchmarking system GUI

The system has two classes. One is called `benchmark-object`, and the other is `present-object`. The class of `benchmark-object` has 6 slots corresponding to the needs of the benchmarking system. Below is a piece of code to define this class.

```
(defclass benchmark-object ()
   ((%compiler-name :initarg :compiler-name
                    :accessor compiler-name)
    (%compiler-version :initarg :compiler-version
                       :accessor compiler-version)
    (%architecture :initarg :architecture
                   :accessor architecture)
    (%benchmark-algorithm :initarg :benchmark-algorithm
                          :accessor benchmark-algorithm)
    (%run-step :initarg :run-step :accessor run-step)
    (%date :initarg :date :accessor date)
    (%results :initarg :results :accessor results)))
```

When the application loads a database, it needs to read the configuration file for that database first in order to understand the database's structure. Since the system understands all needed information about the database, it loads the database file line by line and parses all those value and stores them with a

benchmark-object object. All of the fields in the database are extracted and put in the correct slot of the object.

Below is an item of the benchmark-object. It has the following slots:

```
%COMPILER-NAME: "sbcl"
%COMPILER-VERSION: "sbcl 1.0.35"
%ARCHITECTURE: "Linux 2.6.32-24-generic i686"
%BENCHMARK-ALGORITHM: ("NIL" "u" "NIL" "NIL" "struct" "NIL" "o")
%RUN-STEP: "100000000"
%DATE: "3487424327"
%RESULTS: ("18.578" "1.413" "1.705" "3.777")
```

The reason the application needs one more class is to distinguish between the database's values and things that are on the GUI application. The author named this class present-object. This class has 3 slots, which are: content, clickedp, and in-group. All the parameters of the benchmarking algorithms, which are displayed on the GUI, are instances of this class.

The content slot stores the value which is used to display on the application. The clickedp slot indicates whether this object is already selected or not. And finally is the in-group slot which indicates the kind of the parameter. It corresponds to the category of the parameters.

```
(defclass present-object ()
   ((%content :initarg :content :accessor content)
    (%clickedp :initarg :clickedp :initform nil :accessor clickedp)
    (%in-group :initarg :in-group :initform nil :accessor in-group)))
```

For instance, [Integer] is an instance of type present-object. It has the following slots:

```
%CONTENT: "Integer"
%CLICKEDP: NIL
%IN-GROUP: "Type"
```

As described above about the presentation type, the GUI is attached to 3 presentation types. They are compiler, benchmark-agorithm, and architecture. They are defined with these functions:

```
(define-presentation-type compiler ())
(define-presentation-type benchmark-algorithm ())
(define-presentation-type architecture ())
```

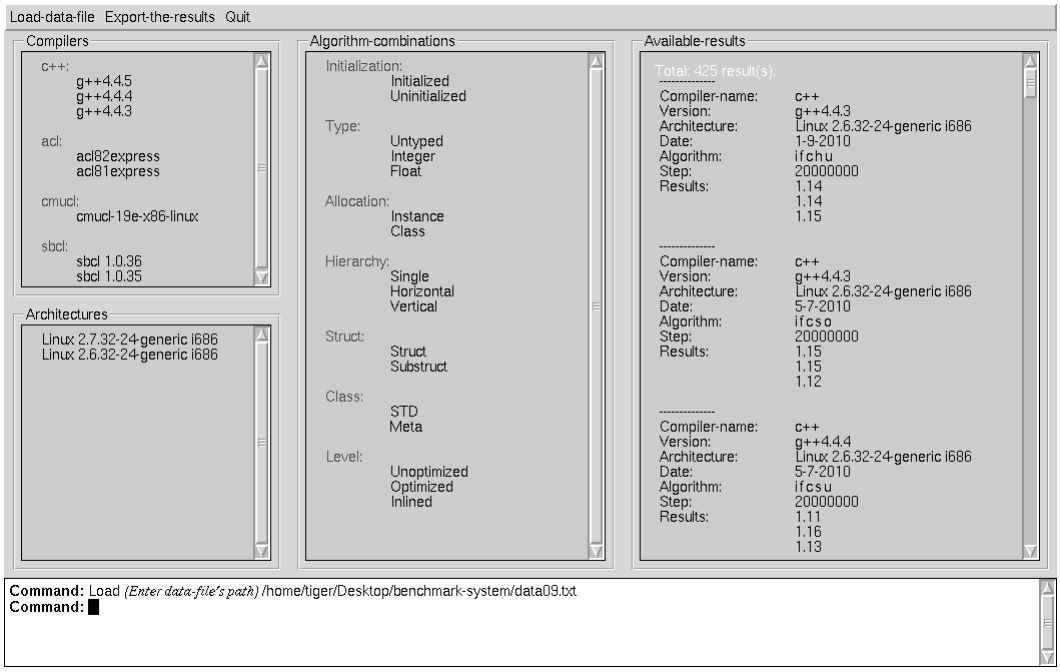

Figure 4: Loading data

Then, all objects that belong to these presentation types execute commands when they are selected from the GUI. Mainly, the application executes the comparison commands to filter the desired values.

6.2.5 The operations

After a database file is loaded, the users can begin to browse the benchmarks by combining all the criteria which are displayed on the application. After a selection, the application filters and updates the results immediately. The users continue doing this until finding the desired benchmarking results.

Not selecting any value in a field is equivalent to selecting all of them.

The GUI application helps the users to visualize the benchmarks easily. The users can track the performance of compilers by both horizontal (across several compilers) and vertical (across several versions of the same compiler).

Figure 5 shows that the users selected `sbcl 1.0.35` for the compilers, the benchmarking algorithms were combined by `Type: Float`, `Dimension: 2` and `Access-type: Pseudo-random`. Then, the application lists all available benchmarks. In this case, there is 2 results for those criteria.

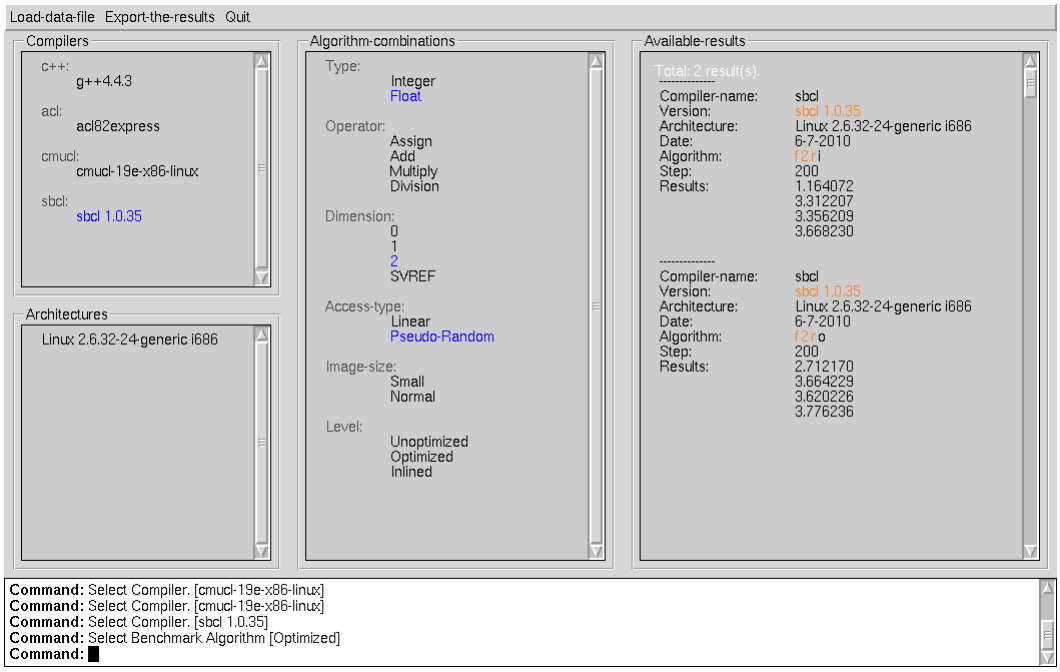

Figure 5: Choosing parameter on GUI

After finding the desired results, the users can save them in a plain text file in order to generate comparision charts with gnuplot. To do this, the users need to select `Export-the-results` from the menu bar, and the application will let the users enter the path where the filtered data will be saved.

7 Future work and conclusions

The benchmarking system for Common Lisp is micro-benchmark. Therefore, the system generates a vast of data files because several compilers are being tested, and several parameters are involved. The new system, which I and Ha Minh Vuong proposed, improved many limitations of the current system so that:

- The new system is more robust. The users can browse, track, and retrieve the benchmarking data easily with the helps of the GUI application.
- The new system is more portable. The operations of the system will not depend on the host systems of the machines:
  - The database is a structured text file, the system does not need a database system to manage.
  - And the system and GUI application is built on Lisp and McCLIM, so they are able to operate on any host machines' platforms.

- The new system is more extensible. It can be used for further benchmarking system in the future, because we have a GUI application can understand the database by reading its configuration file.
- And the new system is more reliable in the meaning of generating comparison charts, by using gnuplot.

This new system has satisfied all of the requirements to become a good benchmarking system for the ongoing research on the behavior and performance of Common Lisp. However, the system could be more convenient for the users to generate graphs. The plan is that the comparison charts would be generated by "clicks" on the GUI. The GUI application will connect to gnuplot and draw the graphs, instead of the users have to type the scripts in the shell command.

The author realises that the performance is one of the important criteria of an application, but the time of the internship has limited the author to make it sufficiently. The author is continuing to clarify the codes, and improve the algorithms in order to make the system's operation, and maintenance be more effective.

Acknowledgements

The author wishes to thank Professor Didier Verna for the instructions; Ha Minh Vuong for the co-operative works; Professor Robert Strandh for the advices and comments; and Daniela Becker, and all of the people in the LRDE, on the IRC #lisp channel for the helps throughout the period of time of the author's studying so that the author is able to accomplish this internship.

References

[1] Froment, J. (2000). MegaWave2 System Library. CMLA, École Normale Supérieure de Cachan, Cachan, France. http://www.cmla.ens-cachan.fr/Cmla/Megawave

[2] Duret-Lutz, A. (2000). Olena: a component based platform for image processing, mixing generic, generative and OO, programming. In Proceedings of the 2nd International Symposium on Generative and Component Based Software Engineering (GCSE) — Young Researchers Workshop; published in "Net.ObjectDays2000", pages 653–659, Erfurt, Germany. http://olena.lrde.epita.fr

[3] Verna, D. (2006). Beating C in scientific computing applications. In Third European LISP Workshop at ECOOP, Nantes, France. http://www.lrde.epita.fr/~didier/research/publis.php

[4] Verna, D. (2009). CLOS Efficiency: Instantiation – On the Behavior and Performance of Lisp, Part 2.1. In Proceedings of the International Lisp Conference , MIT, Cambridge, Massachusetts, USA. http://www.lrde.epita.fr/~didier/research/publis.php

[5] McCLIM manual http://common-lisp.net/project/mcclim/mcclim.pdf and Robert Strandh and Timothy Moore (2002). A Free Implementation of CLIM, http://common-lisp.net/project/mcclim/clim-paper.pdf

A APPENDIX - The User Manual

This user manual is applied to the new benchmarking system for Common Lisp.

A.1 System requirements

- A Common Lisp Compiler. SBCL is recommended.
- McCLIM library.
- Emacs or XEmacs.
- SLIME, to enable the Lisp mode for Emacs/XEmacs.

A.2 The format of the database and the configuration file

A.2.1 The format of the database

A database file should have an extension of .txt.

The special symbols

- Comma , is used to separate fields.
- Sharp symbol # is used to signal a line of comment.

The structure

Records are presented in separated lines. Each record has:

- Compiler name.
- Compile version.
- Architecture of the machine.
- The number of steps.
- The date at which benchmark was run, in universal format.
- Benchmarking algorithm.
- The results. The results follow the benchmarking algorithm, and have several values.

This information is presented as below:

```
Compiler name,compiler version,architecture,steps,date,{benchmarking algorithm},results
```

The format of the database file is:

```
#Comment-1
#Comment-2
Compiler name,compiler version,architecture,steps,date,{benchmarking algorithm},results
#Comment-3
Compiler name,compiler version,architecture,steps,date,{benchmarking algorithm},results
...
```

Notes

- There is no space after a comma.
- {benchmarking algorithm} is also a comma – separated list of sub-fields. Use "NIL" to indicate that a particular sub-field is unavailable in a benchmark.
- We can arrange the order of parameters in {benchmarking algorithm} as we want, but we have to remember this order so that we describe it in the configuration file correctly.
- The results part also has several values. Use "NIL" to indicate that a particular result is not available.

A.2.2 The format of the configuration file

The name of the configuration file must be the same as the database file's. The file extension is .conf.

The special symbols

- Parenthesis () are used to define a parameter.
- Quotation marks "" are used to define the contents of a parameter.
- Colon : is used to define short name for a parameter.
- Space is used between the contents of a parameter to separate them.
- Sharp symbol # is used to write a line of comment.

The structure

To write a configuration file, please remember the order of the parameters, which were organized in the {benchmarking algorithm} in database file.

- All parameters are presented in separated lines.
- The order of lines expresses the orders of parameters in the {benchmarking algorithm} in the database.

The format of the configuration file is:

```
#Comment-1
#Comment-2
(“Parameter-1” “Value-1:v1” “Value-2:v2”)
(“Parameter-2” “Value-1” “Value-2” “Value-3:v3”)
#Comment-3
(“Parameter-3” “Value-1” “Value-2:va-2” “Value-3:v3”)
...
```

A.2.3 Example

We take ’09 system to make the ideas more obvious.

The configuration file of ’09 system is:

```
("Initialization" "Initialized:i" "Uninitialized:u")
("Type" "Untyped:u" "Integer:i" "Float:f")
("Allocation" "Instance:i" "Class:c")
("Hierarchy" "Single:s" "Horizontal:h" "Vertical:v")
("Struct" "Struct:struct" "Substruct:substruct")
("Class" "STD:std" "Meta:meta")
("Level" "Unoptimized:u" "Optimized:o" "Inlined:i")
```

Below is one record which is extracted from ’09 database file:

```
sbcl,sbcl 1.0.35,Linux 2.6.32-24-generic
i686,25000000,3487424498,NIL,f,i,s,NIL,std,i,NIL,1.265,2.581,12.325
```

- sbcl: compiler name.
- sbcl 1.0.35: compiler version.
- Linux 2.6.32-24-generic i686: architecture of the machine.
- 25000000: number of steps. 3487424498: the date when we executed the benchmark.
- NIL,f,i,s,NIL,std,i: the benchmarking algorithm.We have 7 parameters to form algorithms in ’09 system. Therefore, we can see in the configuration file above, we have 7 lines to define these parameters. In this case:
  - NIL: no value for Initialization.
  - f: Float - Type.
  - i: Instance - Allocation.
  - s: Single - Hierarchy.
  - NIL: no value for Struct.
  - std: STD - Class.

  - i: Inlined - Level

- NIL,1.265,2.581,12.325: the results of the benchmark. The first result is not available, its value is NIL.
  - NIL: no value for "No slot" case.
  - 1.265: the value of "1 slot" case.
  - 2.581: the value of "7 slots' case.
  - 12.325: the value of "49 slots" case.

A.3 Operate the system

- Make the ASDF system recognize our system by making a symbolic link of "benchmark-system.asd" in the home folder (Unix system).
- Start Emacs and go into Lisp mode by using SLIME: at the screen of Emacs, press `Alt+x`, type `slime` and press `Return`.
- Type `(asdf:oos 'asdf:load-op :benchmark-system)`.
- After the system loaded, we change into the package of the system: `(in-package :benchmark-system)`. We can see the changing between packages by the string "CL_USER" is changed to "BENCHMARK_SYSTEM".
- At this point, we can choose to make a database file from a benchmarking system, or start the GUI application.

A.3.1 Make the database file

To create (collect all data of '06 system exactly) database for '06 system, from the scratch:

- Run the `(write-comment06)` to create a file with comments. If the file exists, it will be overwritten. If it does not, it will be created. It just writes the comments to a file.
- Then run the `(make-database06)` to create database. It will collect all data of the '06 benchmarking system to the database file. The data will be appended to the existed one (the file with comments). If you want to save the data of the same benchmarking system in a different file after each time benchmarking, just change the path-name of this file in `*db06*` variable, and run two commands above. It is the same for '09 system.

Notes

One more thing to consider, all the available compilers of the '06 and'09 system and its version are stored in `*impl*` variable in "data.lisp". The system can be benchmarked with another version of those compilers. Thus, to have the correct data in database, please check the versions of your system with this variable, and modify it if needed.

A.3.2 Start the application

- Type `(benchmark-system-gui)` and press `Return`.

  After the application started, we can load a database to start browsing the benchmarks. There are 2 ways to choose a database:

  - The first option is to click on the menu-bar and choose `Load-data-file`.
  - The second option is to type the command `load` in the interaction-pane.

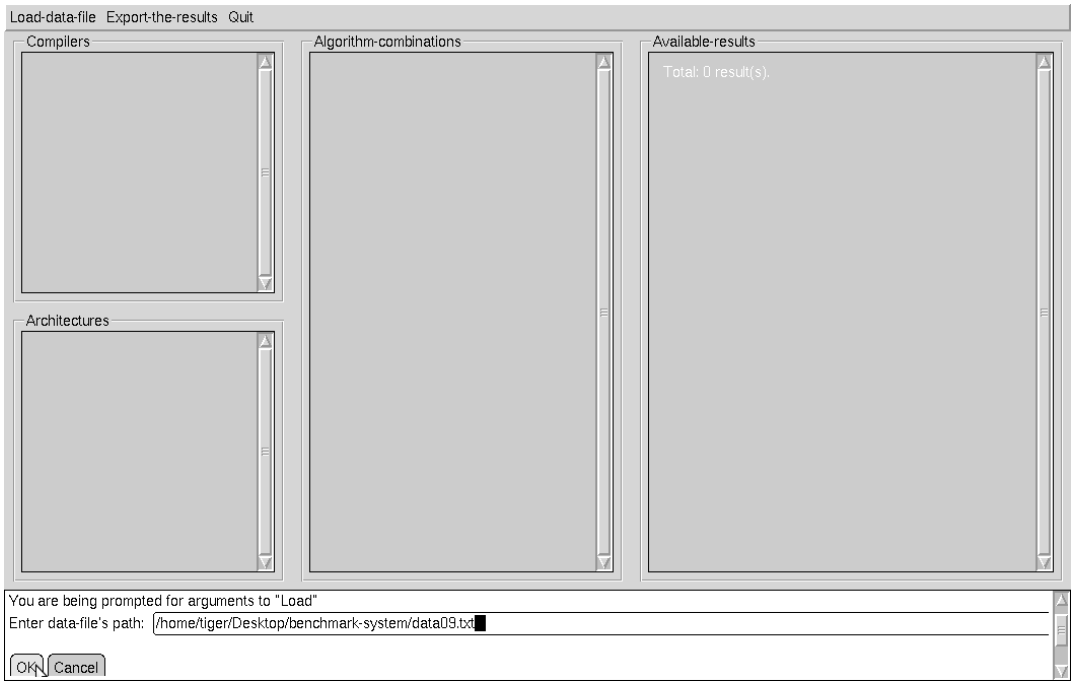

Figure 6: The Graphical User Interface of the application

Now we click on `Load-data-file` on the menu-bar. In the interaction-pane, the field for inputting appears. We can type the path where the application can find the database and its configuration file.

Figure 7 is the interface of the application when a database is loaded. All of the parameters are displayed for users to select, and the rightmost pane displays the available benchmarking results.

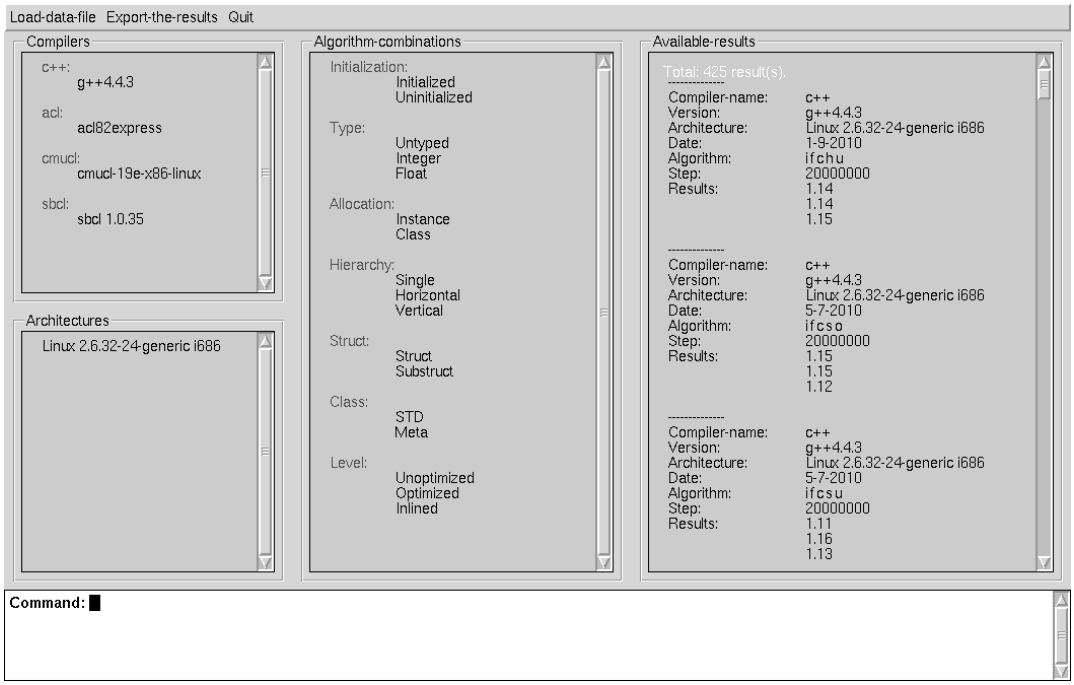

Figure 7: The database is presented on the application

Now, we can start browsing all the benchmarking records by combining all available parameters. When a parameter is clicked, it and its corresponding value in the benchmarking record will be colored in blue and orange for us to be easy to track; and the results are updated.

After finding the desired results, we can export them to a plain text file in order to make comparison charts with gnuplot. To export the data, we choose `Export-the-results`, then, in the interaction-pane, the inputting field appears for us to type the file name and the path where we want to save it.

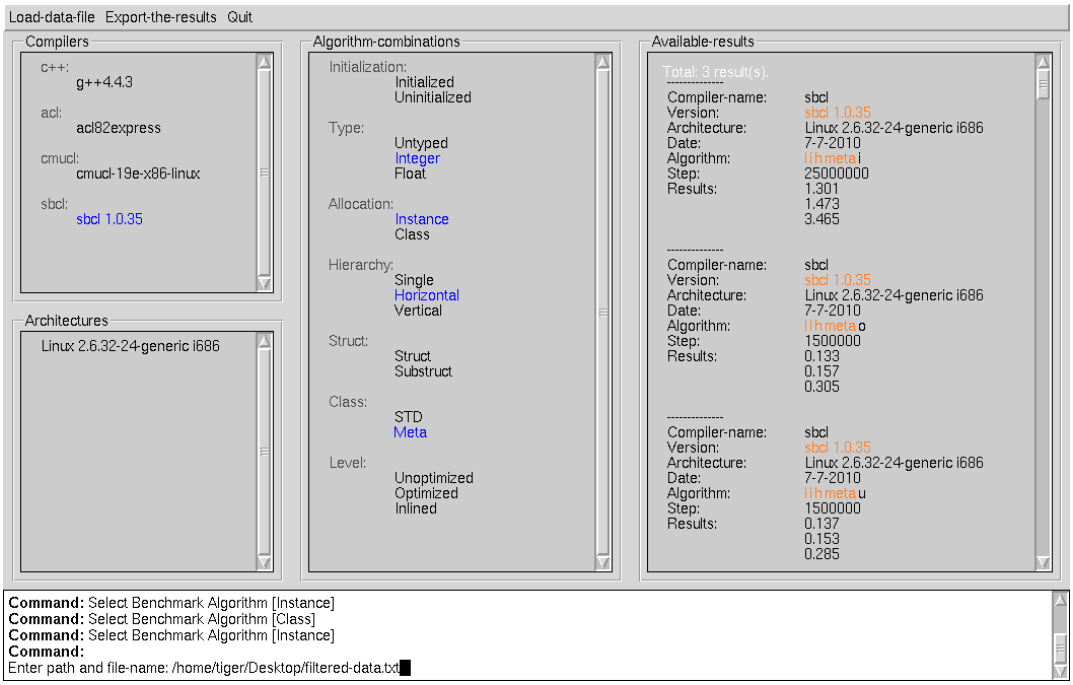

Figure 8: Export filtered data

If we would like to change to another database, we can choose `Load-data-file` as instruction above, or we can type `load` in the interaction-pane, then follow with a `Space`.

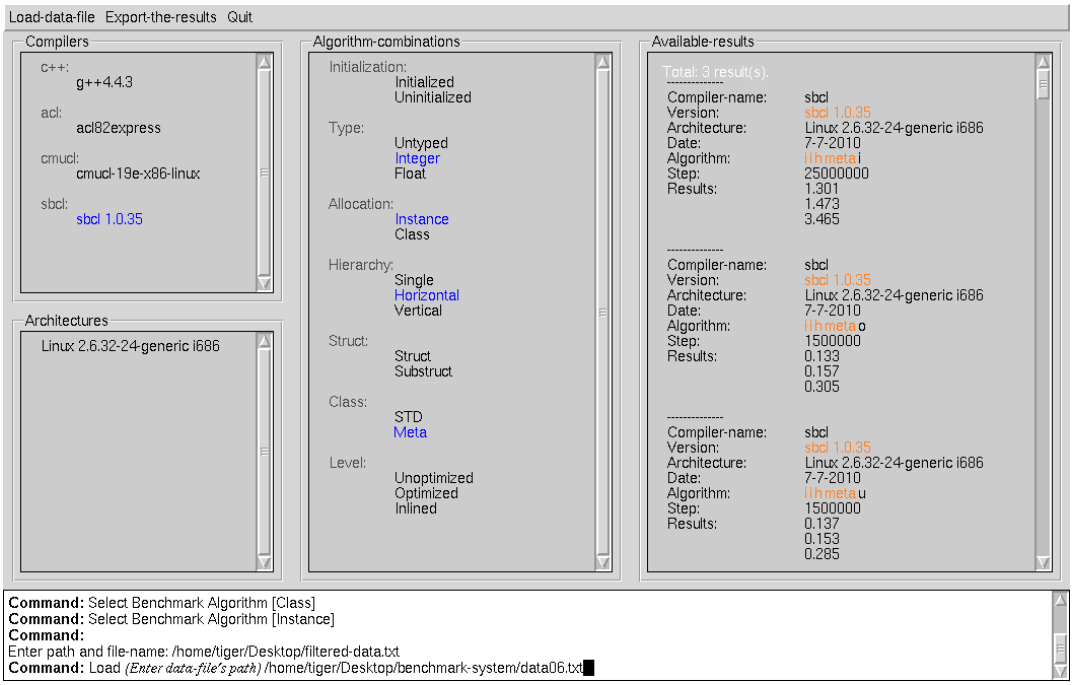

Figure 9: Load database with command `Load`

Then, the interaction-pane allows us to input the path of the database file. After typing in the database's path, we hit `Space` one more time. Then the application loads the new database and displays it on the GUI.

A.4 The application's warnings

A.4.1 The "empty lines in database/configuration files" warning

If the database or the configuration file has some empty lines inside, the application still works well. But it gives the notification about that, and lines' positions where we can look for in case we would like to fix. The picture 10 is in this situation.

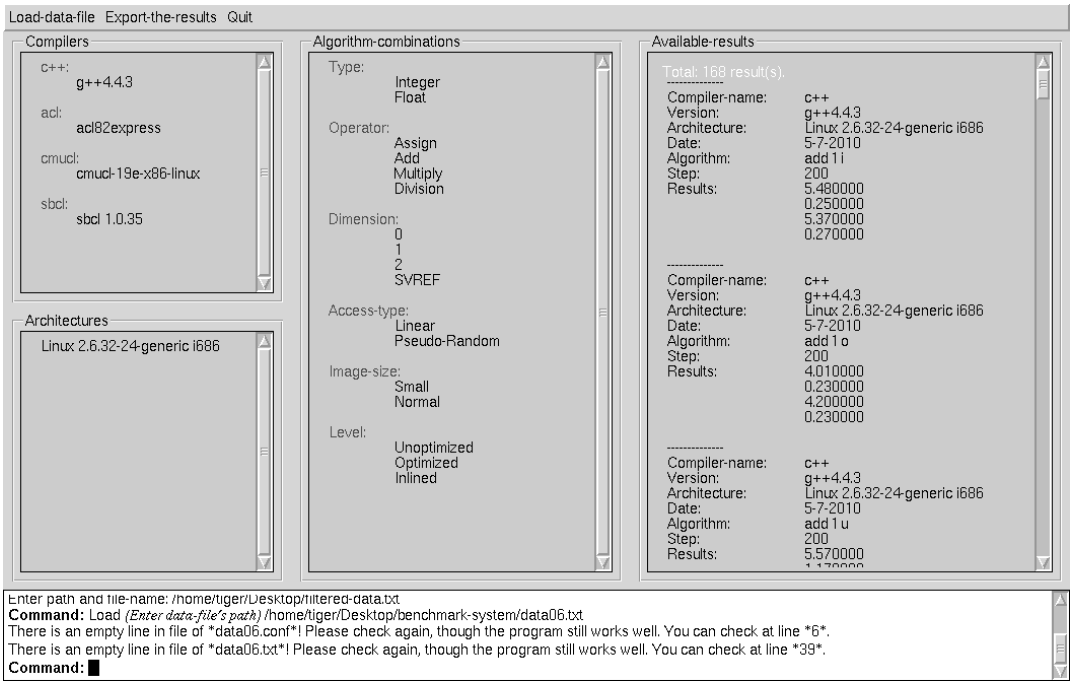

Figure 10: The warnings when loading database/configuration files with empty lines inside

A.4.2 The "database file not found" error

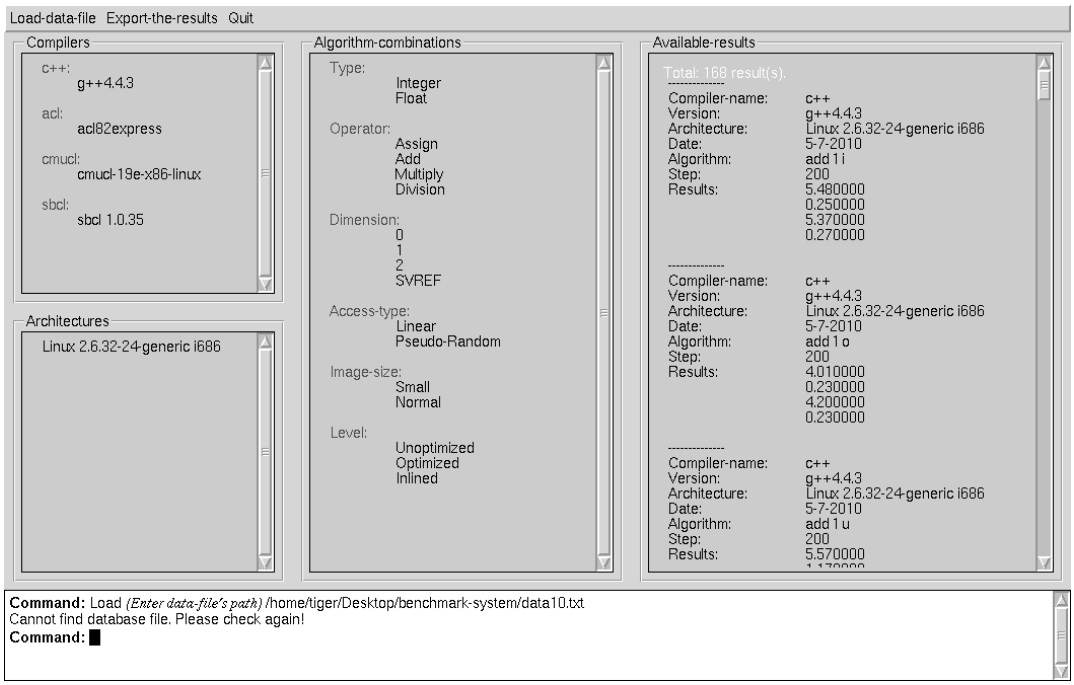

Figure 11: The error when database file not found

If the application cannot find the database with the given path, it will give a message "Cannot find database file. Please check again!" likes the picture above.

A.4.3 The "database's configuration file not found" error

If the corresponding configuration file of the given database cannot be found, the application will give a message that it cannot find the configuration file, and it gives a suggestion is that which name of the configuration file should be.

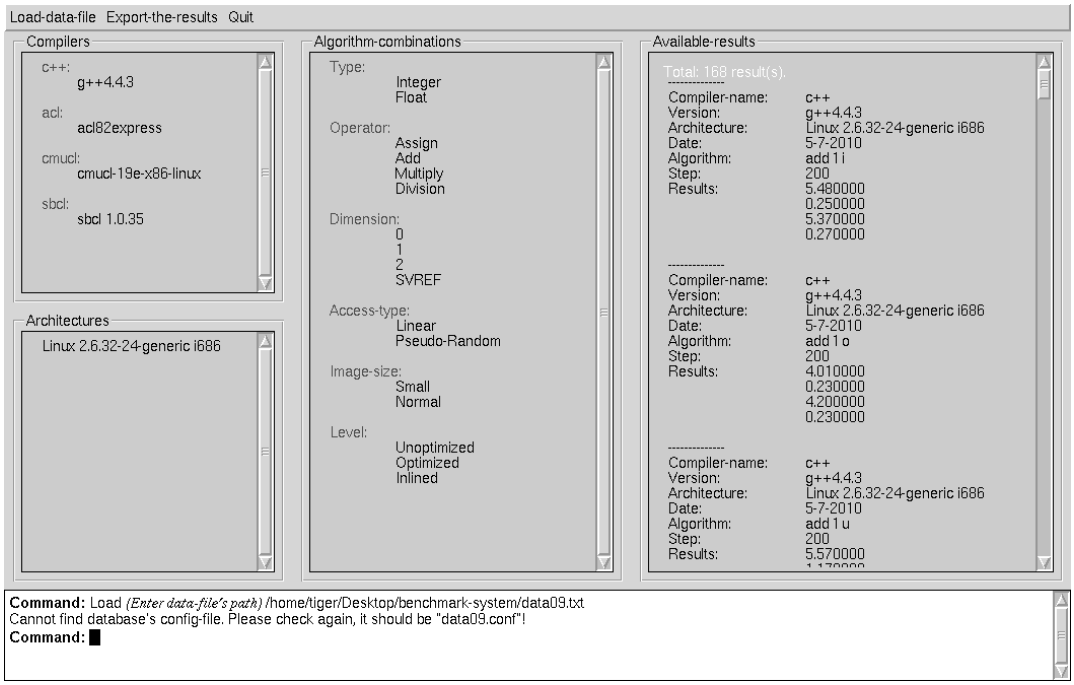

Figure 12: The error when the corresponding configuration file of the database not found.