Multicube Explorer User Manual
Version 0.5

Politecnico di Milano (*Italy*) and Università della Svizzera italiana (*Switzerland*)

January 12, 2009

Contents

1 Overview of Multicube Explorer

Multicube explorer is an interactive program that lets the designer explore a design space of configurations for a parameterized architecture for which an executable model (use case simulator) exists. Multicube explorer is an advanced multi-objective optimization framework which is entirely command-line/script driven and can be retargeted to any configurable platform by writing a suitable XML design space definition file and providing a configurable simulator. Multicube Explorer is supported by the EC under grant FP7-216693 MULTICUBE (http://www.multicube.eu). The tool and the documentation can be currently found at the following address: http://home.dei.polimi.it/zaccaria/multicube_explorer.

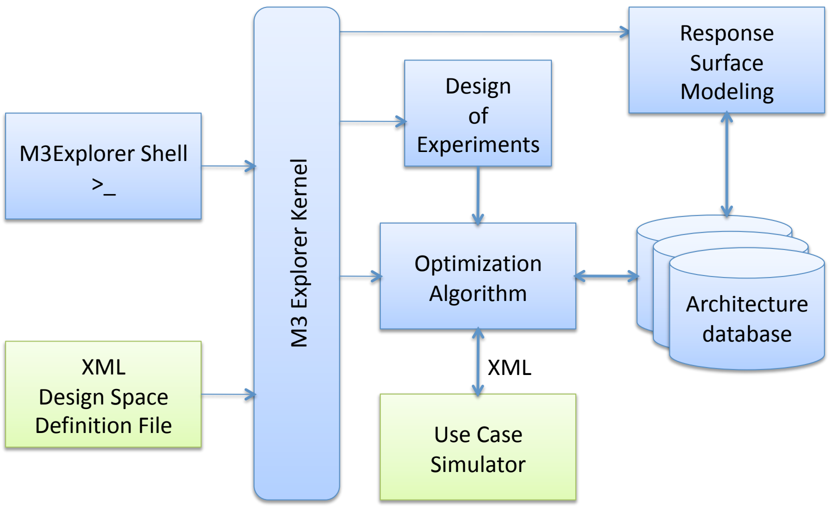

Figure 1: Structure of Multicube Explorer

1.1 Goals of Multicube Explorer

The overall goal of the open source design space exploration framework aims at providing a retargetable tool to drive the designer towards near-optimal solutions to the architectural exploration problem, with the given multiple constraints. The final product of the framework is a Pareto curve of configurations within the design evaluation space of the given architecture.

1.1.1 Automatic design space exploration

One of the goals of the open source tool is to provide a command line interface to the exploration kernel that allows the construction of automated exploration strategies. Those strategies are implemented by means of command scripts interpreted by the tool without the need of manual intervention. This structure can easily support the batch execution of complex strategies that are less prone to human intervention, due to their execution time.

1.1.2 Portability

Another goal of the open source tool is to be portable across a wide range of systems. This goal will be achieved by not sacrificing the efficiency of the overall exploration engine. The standard ANSI C++ programming language will be used for developing the open source framework. The Standard Template Library as well as other open source libraries will be used during the development process.

1.1.3 Modular composition

One of the strength of the open-source tool is the modularity of its components. Simulator, optimization algorithms and other design space exploration components are dynamically linked at run-time, without the need of recompiling the entire code base. This will be supported by well-defined interfaces between the drivers supporting the simulation and the optimization algorithm. This will strongly enable the introduction of new modules for both academic and industrial purposes. Given the modular decomposition, a single optimization algorithm can used for every use case simulator. Moreover, a single use case architecture can be optimized with a wide range of optimization algorithms.

1.2 Architecture of the tool

The tool (Figure 2) is basically composed by an exploration kernel which orchestrates the functional behavior of the design of experiments and optimization algorithms.

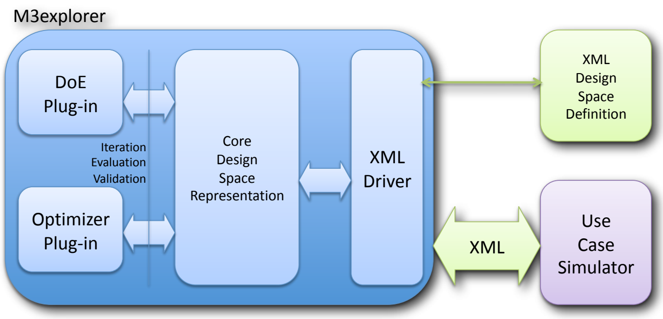
M3explorer
DoE Plug-in
Iteration Evaluation Validation
Optimizer Plug-in
Core Design Space Representation
XML Driver
XML
XML Design Space Definition
Use Case Simulator

Figure 2: Architecture of the tool

The kernel module is responsible for reading in the design space definition file (in XML format) and accepting commands from the shell interface (or the corresponding script). It then exposes the parameters of the design space to all the modules involved in the optimization process (DoE, Optimization Algorithms) by means of a core design space representation.
The core design space representation provides a set of abstract operations that are mapped on the specific use case under analysis. The abstract operations are represented by iterators over the feasible design space, among which we can find:

- Full search iterators.
- Random search iterators, (global and neighborhood).
- Factorial iterators (two-level, two-level + center point).

The core design space representation provides also services for validating architectural choices at the optimizer level and evaluating the associated objective functions. The objective functions are defined as a subset of the use case system level metrics and can be manipulated by the user by interacting with M3explorer.

1.3 Interaction with the simulator

The design space exploration is performed by using the simulation abstraction layer exported by the XML driver to the optimizer plug-ins. In principle, the optimizer instantiates a set of architectural configurations by means of the design space iterators, and passes the corresponding representation to the XML driver which will execute the simulator (see Figure 3). Information about simulator runs will be displayed directly on the M3explorer shell.

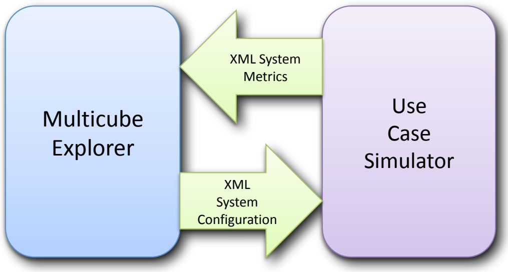

Figure 3: Interaction of Multicube Explorer with the Use Case Simulator

M3Explorer creates a specific directory to execute each instance of the simulator. In this directory, a valid system parameters file is created before starting the simulator. A system metrics file is expected to be obtained as the output of the simulator execution.

2 License

Multicube explorer is open-source and it is released under the BSD license:

3 Installation Requirements and Procedure

3.1 Installation Requirements

Multicube Explorer (*m3explorer* in short) has been designed to be compatible with LINUX and BSD (MAC-OSX) platforms. M3Explorer is written in C++ and can be compiled with a standard GNU C++ compiler (version 4.0.1 or higher).
The following libraries and programs are needed for correctly compiling the software:

- The Bison/YACC parser generator (http://www.gnu.org/software/bison/bison.html)
- The Flex/Lex lexical analyzer (http://flex.sourceforge.net/)
- The libxml2 library and libxml2-devel (The latest versions of libxml2 can be found on the xmlsoft.org server)
- The Gnuplot software.

In most cases (including common Linux distributions), these tools are already available on the system.

3.2 Installation Procedure

M3Explorer is distributed in source form. In order to be executed, it must be compiled and installed into a standard directory.

- Download the compressed file containing the release of M3Explorer (.tar.gz). The current release of the tool is 0.5.
- Uncompress the downloaded archive:

  ```
  > tar zxvf m3explorer_release_0_5.tgz
  ```

  This will create a directory m3explorer; the complete path-name to this directory will be referred to as `sourcedir`.
- Create a `build` dir, and run the `configure` command into it:

  ```
  > mkdir build
  > cd build
  > <sourcedir>/configure --image=<installdir>
  ```

  where `installdir` is the final installation directory of the software. If `installdir` is not specified, the software will be installed in `./image`
- Run `make` and `make install` to finish the installation.

  ```
  > make
  > make install
  ```

- (Optional) To delete all the temporary files generated during the compilation, use the following command:

  ```
  > make dist-clean
  ```

  This command deletes everything that has been built with `configure` and `make`.

3.3 Testing the installation

To test that the installation has been performed correctly, first run the application executable `m3explorer` which is present in the installation dir:

```
> <installdir>/bin/m3explorer


         ----              _
   _ __ |__ / _____ ___ __| |___ _ _ ___ _ _
  | '  \ |_ \/ -_) \ / '_ \ / _ \ '_/ -_) '_|
  |_|_|_|___/\___/_\_\ .__/_\___/_| \___|_|
                     |_|

  Multicube Explorer - Version release_0_5
  Send bug reports to zaccaria@elet.polimi.it, gpalermo@elet.polimi.it
  --

m3_shell>
```

To exit the program, just type `exit` followed by a return.
A more comprehensive test can be run with the `do_test` script which is present in the `test` directory of the installation image (<installdir>). A succesfull execution produces the following output:

```
$ <installdir>/tests/do_tests
1) Tested XML design space construction: PASSED
2) Tested XML rules                    : PASSED
3) Tested XML error reporting          : PASSED
4) Tested XML input/output interface   : PASSED
5) Tested full factorial features      : PASSED
```

3.4 Uninstall procedure

If you installed m3explorer into a dedicated directory, it can be removed by simply deleting the directory. Otherwise, the specific executable and the associated installation files should be removed manually.

3.5 Documentation

The documentation of M3Explorer is mainly composed by two documents:

- The present *User Guide*.
- The *Developer Guide*. This guide can be browsed either on the M3Explorer website or generated by means of the doxygen document production system (use `make doc` to generate the guide in the installation image (<installdir>).

4 The Shell of Multicube Explorer

To run M3Explorer you need to launch the following command:

```
> <installdir>/bin/m3explorer


         ____              _
   _ __ |__ / _____ ___ __| |___ _ _ ___ _ _
  | '  \ |_ \/ -_) \ / '_ \ / _ \ '_/ -_) '_|
  |_|_|_|___/\___/_\_\ .__/_\___/_| \___|_|
                     |_|

  Multicube Explorer - Version release_0_5
  Send bug reports to zaccaria@elet.polimi.it, gpalermo@elet.polimi.it
  --

m3_shell>
```

and the M3Explorer Shell for the user interaction starts. To exit the program, just type `exit` followed by a return.

Typing the `help` command all the available M3Explorer commands with short descriptions are shown.

```
m3_shell> help
db_change_current               change the current database
db_export                       exports the db into a csv file
db_filter_pareto                filters the current database for pareto points
db_insert_point                 insert a point in the current database
db_read                         reads a database from disk
db_report                       reports the contents of a database
db_show_optimum                 shows the optimal point of a database
db_write                        writes a database on disk
doe_define_doe                  define the doe module
drv_define_driver               define the driver module
drv_show_info                   shows information about current driver
exit                            quit the current m3explorer session
help                            general help on m3explorer commands
opt_define_optimizer            define the optimizer module
opt_show_info                   shows information about current optimizer
opt_tune                        start the exploration process
quit                            quit the current m3explorer session
read_script                     read script from file
set                             set a variable to a specific value
show_vars                       shows the variables in the current shell
m3_shell>
```

The design space exploration problem within M3Explorer, is defined by a driver. An XML driver, `m3_xml_driver`, is provided in M3Explorer. Such driver allow the integration of M3Explorer with other performance estimation tools by exploiting well defined XML interfaces that will be described later in section 6.
Other two drivers, `m3_dtlz_driver` and `m3_test_driver` are distributed in the purpose of methodological test of new design space exploration techniques.
To load the description of the design space for the following exploration process, M3Explorer should be lauched with the `-x <design_space_file>.xml` flag.

E.g. `$ <installdir>/bin/m3explorer -x simple_sim_ds.xml`

The use of M3Explorer can be both in interaction mode through the shell or in a script mode.
This second mode can be enabled by writing all the M3Explorer commands into a script file and then launching M3Explorer with the `-f <M3Explorer_commands_file>.scr` flag.

```
$ <installdir>/bin/m3explorer -x simple_sim_ds.xml -f simple_sim_scr.scr
```

5 Available Plugins

5.1 DoEs

The term Design of Experiments (DoE) is used to identify the planning of an information-gathering experimentation campaign where a set of variable parameters can be tuned. The reason for DoEs is that very often the designer is interested in the effects of some parameter's tuning on the system response. Design of experiments is a discipline that has very broad application across natural and social sciences and encompasses a set of techniques whose main goal is the screening and analysis of the system behavior with a small number of simulations. Each DoE plan differs in terms of the *layout* of the selected design points in the design space.
The available DoEs in the M3Explorer framework are:

- Full Search
- Random
- Two Levels Full Factorial

5.1.1 Full Search

It is the simplest DoE in the discrete world. It consider all the possible configuration of the design space.

```
m3_shell> doe_define_doe "m3_full_doe"
```

5.1.2 Random

The design space configurations are picked up randomly by following a Probability Density Function (PDF). In the implemented plugin uses a uniformly distributed PDF.

```
m3_shell> doe_define_doe "m3_random_doe"
```

The variable `solutions_number` can be used to define the number of points of the random DoE. e.g. for a random doe with 15 points

```
m3_shell> set solutions_number = 15
```

5.1.3 Two levels Full Factorial

In statistics, a factorial experiment is an experiment whose design consists of two or more parameters, each with discrete possible values or ”levels”, and whose experimental units take on all possible combinations of these levels across all such parameters. Such an experiment allows studying the effects of each parameter on the response variable, as well as the effects of interactions between parameters on the response variable. In this plugin, we consider a 2-level full factorial DoE, where the only levels considered are the minimum and maximum for each parameter.

```
m3_shell> doe_define_doe "m3_two_level_ff"
```

5.2 Optimizers

The available optimizer plugins in M3Explorer are:

- Pareto DoE

- APRS: Adaptive windows Pareto Random Search
- MOSA: Multi-Objective Simulated Annealing
- MOPSO: Multi-Objective Particle Swarm Optimizer
- NSGA-II: Non-dominated Sorting Genetic Algorithm

5.2.1 Pareto DoE

This is not a real optimizer but it is only a simple method used to evaluate the point selected by the DoE.

```
m3_shell> opt_define_optimizer "m3_pareto_doe"
```

5.2.2 APRS

This algorithm called Adaptive windows Pareto Random Search is an algorithm that has a dynamic windows size which are reduced with the time spent in the exploration and with the goodness of the point found in the current windows. The windows are centered on the current pareto solutions and the new configurations are randomly selected within the windows.

```
m3_shell> opt_define_optimizer "m3_aprs"
```

5.2.3 MOSA

Simulated annealing is a Monte Carlo approach for minimizing multivariate functions. The term simulated annealing derives from the analogy with the physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure. In the Simulated Annealing algorithm a new configuration is constructed by imposing a random displacement. If the cost function of this new state is less than the previous one, the change is accepted unconditionally and the system is updated. If the cost function is greater, the new configuration is accepted probabilistically; the acceptance possibility decreases with the temperature (optimization time). This procedure allows the system to move consistently towards lower cost function states, thus 'jumping' out of local minima due to the probabilistic acceptance of some upward moves.
This optimizers implemented in M3Explorer is called Multi-Objective Simulated Annealing (MOSA) and it is derived by: *Smith, K. I.; Everson, R. M.; Fieldsend, J. E.; Murphy, C.; Misra, R., "Dominance-Based Multiobjective Simulated Annealing",IEEE Transaction on Evolutionary Computation, 12(3): 323-342 - 2008*

```
m3_shell> opt_define_optimizer "m3_mosa"
```

5.2.4 MOPSO

More in general, Particle Swarm Optimization (PSO) is a heuristic search methodology that tries to mimic the movements of a flock of birds aiming at finding food. PSO is based on a population of particles flying through an hyper-dimensional search space. Each particle possesses a position and a velocity; both variables are changed to emulate the social-psychological tendency to mimic the success of other individuals in the population (also called *swarm*).
This optimizers implemented in M3Explorer is called Multi-Objective Particle Swarm Optimization (MOPSO) and it is derived by: *G. Palermo, C. Silvano, V. Zaccaria. "Discrete Particle Swarm Optimization for Multi-objective Design Space Exploration", In Euromicro Proceedings of DSD'08 - Conference on Digital System Design. September 2008*

```
m3_shell> opt_define_optimizer "m3_mopso"
```

5.2.5 NSGA-II

In a Genetic Algorithm, many design alternatives belonging to design space are seen like individuals in a stored population. The exploration procedure consists of the simulation of the evolution process of generation of individuals and the improvement of solutions belonging to next generations is explained by Darwinian theory. The evolutionary operators describe how individuals are selected to reproduce, how a new generation of individuals is generated from parents by crossover and mutation and how new generation of individuals is inserted into population replacing or not the parents.
The implemented approach for Multiobjective optimization is the non-dominated sorting genetic algorithm (NSGA-II) described in: *Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, 2002*

```
m3_shell> opt_define_optimizer "m3_nsga_II"
```

6 Interfaces for the integration of new simulators

This section describe how it is possible to integrate M3Explorer with a system simulator and how to define to M3Explorer the design space to explore.

6.1 Design Space Definition

The definition of the design space is done by using an XML file that is composed of a preamble, which defines the namespace and a version, which, in this document, is 1.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<design_space xmlns="http://www.multicube.eu/" version="1.3">
        <simulator> ... </simulator>
        <parameters> ... </parameters>
        <system_metrics> ... </system_metrics>
        <rules> ... </rules>
</design_space>
```

The remaining part of the file describes the simulator invocation method (`<simulator> ... </simulator>`), the set of parameters of the simulator which can be configured (`<parameters> ... </parameters>`), the system metrics which can be estimated by the simulator (`<system_metrics> ... </system_metrics>`) and the rules which have to be taken into account by M3Explorer in order to generate feasible configurations.

6.1.1 Simulator Invocation

The `<simulator_executable>` marker is used for specifying the complete path name of the executable:

```
<simulator>
        <simulator_executable path="/path/my_simulator_executable" />
</simulator>
```

The path is specified by using Unix conventions. The simulator executable is invoked with three arguments:

```
my_simulator_executable \
        --xml_system_configuration=sc_path_name \
        --xml_system_metrics=sm_path_name \
        --reference_xsd=xsd_file_name
```

where `sc_path_name` is the path name of XML file describing the system configuration to be passed to the simulator. The `sm_path_name` is the path name of the output XML file which should be used by the simulator for producing the system metrics output. The argument `-reference_xsd=xsd_file_name` is used for specifying the position of the reference M3Explorer/simulator interface XSD file in the file system. This argument can be used by the simulator for validating the input and output files exchanged with M3Explorer.

6.1.2 Parameters Definition

The `<parameters> ... </parameters>` is used by the use case and simulator provider to specify the names, the types and the ranges of the parameters that can be explored by the DSE tool. The section contains a list of <parameter> markers:

```
<parameter>
        <parameter name="seed" description="RNG seed" type="integer" min="0" max="10"/>
        <parameter name="fetch_queue_size"
                   description="instruction fetch queue size"
                   type="integer" min="1" max="8" step="2"/>
        ...
</parameters>
```

For each parameter a unique name must be provided. This name will be used for generating configurations at the input of the simulator. Feasible parameter names are identified by the following regular expression:

[A-Za-z_] [A-Za-z0-9_]*

The parameters types can be divided into two categories:

- Scalar types
- Variable vector types

Scalar parameter types. The scalar parameter type can be:

- `integer` and `boolean`. The integer type specifies a simple sequential integer progression associated that specific parameter. The min and max attributes (which are mandatory) specify the boundaries of the progression. The step attribute can be used to produce non-unitary progressions. The Boolean type is an integer progression with min=0 and max=1.
- `exp2` The values associated with an "exp2" parameter type should be computed by M3Explorer by using a power of two progression. For example:

  ```
  <parameter name="il1_cache_block_size_bytes"
             description="..." type="exp2" min="8" max="64"/>
  ```

  should be interpreted by M3Explorer as a parameter with range values:

  { "8", "16", "32", "64" }

- `string` In the case of string parameters, a list of possible string values should be used instead of the min/max attributes:

  ```
  <parameter name="bpred" description="branch predictor type" type="string">
          <item value="nottaken"/>
          <item value="taken"/>
          <item value="perfect"/>
          <item value="bimod"/>
          <item value="2lev"/>
          <item value="comb"/>
  </parameter>
  ```

Variable vector types. The following types are introduced for producing variable vector types with specific constraints on the possible combinations of the components:

- `on-off mask`. The on-off mask is essentially a vector combination of boolean values with a specific dimension. We use the `on_set_size` attribute to specify the amount of elements which should be "on" in the resulting vector:

  ```
  <parameter type="on_off_mask"
             name="active_processors"
             dimension="7"
             on_set_size="@number_of_threads" />
  ```

  The on_set_size can be a fixed value or a reference to a variable value. In the case of reference to variable values, the notation @parameter should be used. For example the notation @number_of_threads indicates that the "on_set_size" should be equal to the "number_of_threads" parameter of the configuration under evaluation. In this example we assume that the "number_of_threads" parameter type is an integer progression without explicit steps; as a matter of fact the notation @_parameter_ can refer only to integer parameters with a step=1. When the on_set_size attribute is not specified, all the possible combinations of the Boolean vector are considered in the generation of the associated progression. The dimension of the on_off_mask can be variable as well:

```
<parameter type="on_off_mask" name="QoS_priorities" dimension="@number_of_threads" />
```

The previous parameter specification contains, as an example, the Boolean QoS priorities for each of the active nodes of a target multi-processor system.

- `Permutation`. Variable size permutations are used, for example, in the case of thread-to-processor mapping problems. In this case a task identifier should be generated for each active processor:

  ```
  <parameter type="permutation" name="thread_assignment" dimension="@number_of_threads" />
  ```

  A permutation contains a non-repeatable sequence of values from 1 to the actual dimension of the vector. For example, the variable vector parameter:

  ```
  <parameter type="permutation" name="example" dimension="2" />
  ```

  can assume the following values [1,2] or [2,1].

6.1.3 System Metrics Definition

The <`system_metrics`> section is used by the use case and simulator provider to specify the names, the types and the units of the system metrics that can be estimated by the simulator:

```
<system_metrics>
        <system_metric name="cycles" type="integer" unit="cycles" desired="small"/>
        <system_metric name="instructions" type="integer" unit="insts" description="..."/>
        <system_metric name="powerconsumption" type="float" unit="W" description="..." />
        <system_metric name="area" type="float" unit="mm2" desired="small" />
</system_metrics>
```

Feasible system metric ”name” attributes are identified by the following regular expression:

[A-Za-z_] [A-Za-z0-9_]*

The optional ”description” attribute is a generic string describing the nature of the system metric.
The ”desired” attribute indicates whether it is desirable to have a ”small”/”big” value of a specific system metric. This attribute is optional and may be taken into account by M3Explorer in the formalization of the optimization problem which, however, it is not part of this specification. M3Explorer expects to find the system metrics defined in this section in the output file of the simulator. The output file name of the simulator is the second argument passed to the simulator executable file.

6.1.4 Feasibility rules

The <`rules`> section is used by M3Explorer in order to not generate invalid or not feasible solutions during the automated exploration process. The behavior of the simulator when these rules are not met is undefined. Each rule is a boolean expression which should evaluate to true for a feasible configuration of the design space. It is up to M3Explorer tool to check for the rules and generate feasible configurations. Each boolean expression can be an operator acting on either a <`parameter`> or <`constant`> leafs or other boolean expressions. This allows creating complex expression trees of rules. Rules are ”AND”ed by default by M3Explorer. Each rule is identified by a <`rule`> marker and it has an optional ”name” attribute. As an example:

```
<rules>
        <rule>
                <greater-equal>
                        <parameter name="l2_cache_block_size"/>
```

```
                        <parameter name="l1_dcache_block_size"/>
                </greater-equal>
        </rule>
        <rule name="application-derived minimal size" >
                <greater-equal>
                        <parameter name="l2_cache_size"/>
                        <constant value="2048"/>
                </greater-equal>
        </rule>
</rules>
```

Describes the rule:
(l2_cache_block_size >= l1_dcache_block_size) AND (l2_cache_size >= 2048)

Available operators. The following operators/markers can be used:

`<greater>, <greater-equal>, <less>, <less-equal>, <equal>, <not-equal>, <expr>`

The `<expr>` marker can be used for introducing generic expressions e.g.:

```
<rule>
        <greater-equal>
                <parameter name="l2_cache_size"/>
                <expr operator="*">
                        <constant value="2"/>
                        <parameter name="l1_cache_size"/>
                </expr>
        </greater-equal>
</rule>
```

The previous set of rules is represents (l2_cache_size >= 2*l1_cache_size). The operators supported by M3Explorer are {+ - * / }.

Combining rules. For combining complex expressions the following markers/operators can be used:

`<and>, <or>, <not>`

For example, the following rules are AND'ed together:

```
<rules>
        <rule name="overall memory subsystem integrity">
        <and>
                <greater-equal>
                        <parameter name="l2_cache_block_size"/>
                        <parameter name="l1_dcache_block_size"/>
                </greater-equal>
                <greater-equal>
                        <parameter name="l2_cache_size"/>
                        <constant value="2048"/>
                </greater-equal>
        </and>
        </rule>
</rules>
```

This corresponds to the following expression:

(l2_cache_block_size >= l1_dcache_block_size) AND (l2_cache_size >= 2048)

If-then-else rule. An ”if(E) then A” predicate is introduced and it is evaluated as:

- TRUE if E is FALSE
- A if E is TRUE

An example for this rule is the following:

```
<rule name="branch prediction design space reduction">
        <if>
                <not-equal>
                        <parameter name="bpred"/>
                        <constant value="bimod"/>
                </not-equal>
                <then>
                        <equal>
                                <parameter name="bpred_bmod_size"/>
                                <constant value="0"/>
                        </equal>
                </then>
        </if>
</rule>
```

This associated predicate expression is:

if(bpred!=bmod) then bpred_bmod_size=0

This rule forces to generate configurations where if bpred!=bmod then bpred_bmod_size=0. These rules can effectively reduce the overall design space. An ”if(E) then A else B” predicate is introduced and it is evaluated as:

- B if E is FALSE
- A if E is TRUE

6.2 M3Explorer/Simulator Interface

The M3Explorer/Simulator interface is composed by 2 files one in output from M3Explorer to the simulator the other one in the opposite direction.

6.2.1 Simulator input file

The simulator input file should contain a preamble and a sequence of `<parameter>` sections where, for each parameter, the name and the value is specified:

```
<?xml version="1.0" encoding="UTF-8"?>
<simulator_input_interface xmlns="http://www.multicube.eu/" version="1.3">
        <parameter name="seed" value="1" />
        ...
</simulator_input_interface>
```

The number of `<parameter>` sections and the name of the parameters should be the same as defined in the XML Design Space description file. The value of the each parameter section should correspond to one of the possible values as defined in the XML Design Space description file. Concerning variable vector parameters, the actual parameter instances are specified with an itemized list. For example, an on_off_mask instance value for the ”active_processors” parameter is described in the simulator input file as the following list:

```
<parameter name="active_processors" >
        <item index="1" value="0" />
        <item index="2" value="1" />
        <item index="3" value="0" />
        <item index="4" value="1" />
</parameter>
```

In the case of a permutation vector, the index attribute is substituted with the position attribute:

```
<parameter name="thread_assignment" >
        <item position="1" value="2" />
        <item position="2" value="3" />
        <item position="3" value="1" />
</parameter>
```

Index and position attributes start from 1 up to the dimension associated to the variable vector.

6.2.2 Simulator Output File

The simulator output file contains a preamble and a sequence of `<system_metric>` sections where, for each metric, the name and the value is specified:

```
<?xml version="1.0" encoding="UTF-8"?>
<simulator_output_interface xmlns="http://www.multicube.eu/" version="1.3">
        <system_metric name="cycles" value="3000" />
        <system_metric name="instructions" value="1500" />
        <system_metric name="power\_consumption" value="2.5" />
        <system_metric name="area" value="25" />
</simulator_output_interface>
```

The number of `<system_metric>` sections and the name of the system metrics should be the same as defined in the XML Design Space description file.

6.2.3 Simulator Error Management

In the case of errors during the simulator execution, the simulator output file should contain a single `<error>` marker indicating the error reason:

```
<?xml version="1.0" encoding="UTF-8"?>
<simulator_output_interface xmlns="http://www.multicube.eu/" version="1.3">
        <error reason="memory-full" kind="fatal"/>
</simulator_output_interface>
```

The attribute reason is a generic string that can contain a report about the error cause. Overall, the error strings of the simulator are meant to be related to:

- memory-full or disk-full problems
- file system permissions problems.
- license problems.
- internal exceptions.
- other.
- consistency or feasibility violation (if checked by the simulator)

The kind can be ”fatal”/”non-fatal”. Fatal errors should block the overall exploration process while non-fatal errors force M3Explorer to skip to the next configuration. If an `<error>` marker is present in the output file, `<system_metric>` markers are ignored by M3Explorer.

7 Example of exploration with a simple simulator

In this section we report a simple example usage of m3explorer. The example consists of the exploration of the parameter space of a simple simulator. The files associated with this example can be located in `<installdir>/examples/simple_sim` directory; namely, they correspond to:

- `simple_sim.py`: Python script representing the simulator of the target architecture to be explored.
- `simple_sim_ds.xml`: Design space to be explored (see Figure 4)
- `simple_sim_scr.scr`: m3explorer script file which automates the steps of the exploration.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<design_space xmlns="http://www.multicube.eu/" version="1.3">
        <simulator>
                <simulator_executable
                 path="/usr/bin/python @image@/examples/simple_sim/simple_sim.py" />
        </simulator>
        <parameters>
                <parameter name="par1_exp2" type="exp2" min="1024" max="4096" />
                <parameter name="par2_step1" type="integer" min="1" max="2" step="1"/>
                <parameter name="par3_step2" type="integer" min="1" max="5" step="2"/>
        </parameters>
        <system_metrics>
                <system_metric name="sum" type="integer" unit="cycles" desired="small" />
                <system_metric name="difference" type="integer" unit="mm2" desired="small" />
                <system_metric name="product" type="integer" unit="mW" desired="small" />
        </system_metrics>
        <rules>
                <rule>
                <greater-equal>
                        <parameter name="par3_step2"/>
                        <parameter name="par2_step1"/>
                </greater-equal>
                </rule>
        </rules>
</design_space>
```

Figure 4: `simple_sim_ds.xml`

In this example, we perform a full-search exploration of the design space shown in Figure 4 by filtering the final results for the pareto set. To start with the exploration, we invoke m3explorer with its target design space.

```
> <installdir>/bin/m3explorer -x simple_sim_ds.xml
```

The target design space is now loaded. Now, m3explorer knows where the simulator is and which are the parameters associated with it. Once in the m3explorer shell, we perform the following steps:

- Configure the optimizer to clean the directory at the end of the exploration.

  ```
  m3_shell> set clean_directory_on_exit = "true"
  ```

- Change the current database to a new database called `full_db` (it will be filled by optimizer module with the exploration results).

```
m3_shell> db_change_current "full_db"
```

- Load the `m3_full_doe` DoE. Full-search considers all the possible combination of the parameters.

```
m3_shell>  doe_define_doe "m3_full_doe"
```

- Load `m3_pareto_doe` optimizer (it visits only the solutions defined by the DoE).

```
m3_shell> opt_define_optimizer "m3_pareto_doe"
```

- Start the exploration.

```
m3_shell> opt_tune
```

- Write the results of the exploration in an internal M3Explorer `.db` format and in a standard `csv` format.

```
m3_shell> db_write "my_full.db"
m3_shell> db_write "my_full.csv"
```

- Set up the objective functions of the problem (to enable Pareto filtering of visited points), perform Pareto filtering of the current database (by eliminating dominated points) and report the results.

```
m3_shell> set objectives = { "sum" "difference" "product" }
m3_shell> db_filter_pareto
m3_shell> db_report
```

- Write the Pareto points in internal and csv format.

```
m3_shell> db_write "my_full.db"
m3_shell> db_write "my_full.csv"
```

- Exit from the shell.

```
m3_shell> exit
```

The example can be automated by using an M3Explorer script (`simple_sim_scr.scr`):

```
> <installdir>/bin/m3explorer -x simple_sim_ds.xml -f simple_sim_scr.scr
```

Figure 5 shows the output of such script.

```
$ ../../bin/m3explorer -x simple_sim_ds.xml -f simple_sim_scr.scr

Information: Creating the xml_driver
Information: Loading the xml_driver
Information: Assigned value "true" to clean_directory_on_exit
Information: Changing current DB to: full_db
Information: Database not existing. Creating a new one.
Information: Loading the full search doe
Information: Current doe has been set to 'Full search doe'
Information: Current optimizer has been set to 'Pareto doe optimizer'
Information: Starting with the pareto doe optimization process
Information: Evaluating point: [ par1_exp2=1024 par2_step1=1 par3_step2=1 ]
Information: Evaluating point: [ par1_exp2=2048 par2_step1=1 par3_step2=1 ]
Information: Evaluating point: [ par1_exp2=4096 par2_step1=1 par3_step2=1 ]
Information: Skipping point: [ par1_exp2=1024 par2_step1=2 par3_step2=1 ]
Information: Skipping point: [ par1_exp2=2048 par2_step1=2 par3_step2=1 ]
Information: Skipping point: [ par1_exp2=4096 par2_step1=2 par3_step2=1 ]
Information: Evaluating point: [ par1_exp2=1024 par2_step1=1 par3_step2=3 ]
Information: Evaluating point: [ par1_exp2=2048 par2_step1=1 par3_step2=3 ]
Information: Evaluating point: [ par1_exp2=4096 par2_step1=1 par3_step2=3 ]
Information: Evaluating point: [ par1_exp2=1024 par2_step1=2 par3_step2=3 ]
Information: Evaluating point: [ par1_exp2=2048 par2_step1=2 par3_step2=3 ]
Information: Evaluating point: [ par1_exp2=4096 par2_step1=2 par3_step2=3 ]
Information: Evaluating point: [ par1_exp2=1024 par2_step1=1 par3_step2=5 ]
Information: Evaluating point: [ par1_exp2=2048 par2_step1=1 par3_step2=5 ]
Information: Evaluating point: [ par1_exp2=4096 par2_step1=1 par3_step2=5 ]
Information: Evaluating point: [ par1_exp2=1024 par2_step1=2 par3_step2=5 ]
Information: Evaluating point: [ par1_exp2=2048 par2_step1=2 par3_step2=5 ]
Information: Evaluating point: [ par1_exp2=4096 par2_step1=2 par3_step2=5 ]
Information: Writing the database to disk
Information: Database correctly written
Information: Saving the database in CSV format..
Information: Assigned value { "sum" "difference" "product" } to objectives
Information: Filtering the database for pareto points..
full_db: Current database contents
[ par1_exp2=1024 par2_step1=1 par3_step2=1 ] :   1026  1022  1024
[ par1_exp2=1024 par2_step1=1 par3_step2=3 ] :   1028  1020  3072
[ par1_exp2=1024 par2_step1=2 par3_step2=3 ] :   1029  1019  6144
[ par1_exp2=1024 par2_step1=1 par3_step2=5 ] :   1030  1018  5120
[ par1_exp2=1024 par2_step1=2 par3_step2=5 ] :   1031  1017  10240
Number of points in the DB: 5
Information: Writing the database to disk
Information: Database correctly written
Information: Saving the database in CSV format..
Information: Removing xml_driver
Information: Exiting from Multicube Explorer shell
```

Figure 5: M3Explorer output while it is running `simple_sim_scr.scr`

8 Shell Command List

This section reports a brief survey of commands available in the Multicube Explorer optimization tool.
The commands are organized in three main classes:

- Basic commands, adopted to control the m3explorer environment in terms of defined variables and other basic functionalities
- Plugins commands, which give powerful flexibility to the user on handling the modular structure of M3Explorer.
- Database commands, used to handle one or more database where visited design points are stored.

8.1 Basic Commands

These commands support the user on performing simple operations as load scripts, display help messages and set variables value and close m3explorer.

Shell Command name:	**exit**
Arguments:	-
Options:	-
Description:	Exits from the m3explorer shell.
Example usage:	`> exit`

Shell Command name:	**quit**
Arguments:	-
Options:	-
Description:	Same as exit command.
Example usage:	`> quit`

Shell Command name:	**read_script**
Arguments:	Name of the script to be read - *string*
Options:	-
Description:	All commands within the given script are sequentially executed in the m3explorer environment.
Example usage:	`> read_script "my_script.scr"`

Shell Command name:	**set**
Arguments:	Name of the variable to be set - *identifier.* Value to be inserted into the variable - *object*
Options:	-
Description:	Set an environment variable. Set command cares about definition of the variable, if this was never used before, or modification of variable content if variable was existing.

Example usage:	`> set my_string = "I'm a string"`
	`> set my_number = 13`
	`> set my_list = {$my_string $my_number 0.9233 }`

Shell Command name:	**show_vars**
Arguments:	-
Options:	-
Description:	Shows the variable in the current m3explorer environment and reports basic information about available Databases.
Example usage:	`> show_vars`

Shell Command name:	**help**
Arguments:	-
Options:	**–long** request a list of the available options and arguments for each command.
Description:	Reports general help on m3explorer commands.
Example usage:	`> help -long`

8.2 Plugins Commands

M3Explorer is organized in modular structure and enables the user to dynamically load precompiled plugins within the environment.
These plugins are DoE modules adopted for initial experimental design and optimization modules for the definition of optimization algorithm to be adopted.
Shell variables can be used for passing additional parameters to the modules.

Shell Command name:	**doe_define_doe**
Arguments:	Name of the Design Of Experiments to be loaded - *string*
Options:	-
Description:	Instantiates the Design of Experiments module to be used during the optimization.
Example usage:	`> doe_define_doe "m3_random_doe"`

Shell Command name:	**doe_show_info**
Arguments:	-
Options:	-
Description:	Displays info message from the DoE module currently loaded in the m3explorer environment, if any. If no doe module is defined, then an error message is shown.
Example usage:	`> doe_show_info`

Shell Command name:	**drv_define_driver**
Arguments:	Name of the driver to be loaded - *string*
Options:	-
Description:	Load a driver module into the m3explorer environment. On doing so, design space is defined and parameters representing design parameter boundaries are organized into the m3explorer environment as shell variables.

Example usage:

```
> set xml_design_space_file="m3_mpeg_use_case.xml"
> drv_define_driver m3_xml_driver
> show_vars
Shell variables:
Name Value
___- _________
cbs [ "16" "128" ]
dcs [ "2048" "65536" ]
dcw [ "1" "16" ]
ds_parameters [ "ics" ...  "pn" ]
ics [ "2048" "65536" ]
icw [ "1" "16" ]
iwidth [ "1" "8" ]
l2cs [ "32768" "1048576" ]
l2cw [ "1" "16" ]
pn [ "1" "16" ]

Databases in memory
Name size
__________-
root (available) 0
```

Shell Command name:	**drv_show_info**
Arguments:	-
Options:	-
Description:	Shows information about the current driver loaded.
Example usage:	`> drv_show_info`

Shell Command name:	**opt_define_optimizer**
Arguments:	Name of the optimizer to be loaded - *string*
Options:	-
Description:	Loads the optimizer plug-in.
Example usage:	`> opt_define_optimizer "m3_aprs"`

Shell Command name:	**opt_show_info**
Arguments:	-
Options:	-
Description:	Shows information about the current optimizer.
Example usage:	`> opt_show_info`

Shell Command name:	**opt_tune**
Arguments:	-
Options:	-
Description:	Launch the exploration process defined in the optimizer (also called optimization or tuning operation). During optimization, all modules (optimizer, driver, doe) interacts, thus they should be all loaded into the environment when `opt_tune` is called. With commands in the following example, the aprs optimization process is launched over the mpeg usecase, available in the `example` subdirectory of the installation directory. The DoE adopted as initial experimental design is a random one. Results of optimization process are placed into the current database (root database when the shell is just opened).

Example usage:	`> set xml_design_space_file="m3_mpeg_use_case.xml"`
	`> drv_define_driver "m3_xml_driver"`
	`> doe_define_doe "m3_random_doe"`
	`> opt_define_optimizer "m3_aprs"`
	`> opt_tune`

8.3 Database Commands

Simulation results are stored into database (db) in memory. Databases can be loaded/stored from/to the hard disk, in such a way that is easy to handle simulation data obtained in different working sessions.
Operations that can be performed on databases are various and allow to extrapolate and visualize some fundamental high level information needed to investigate solution quality of multiobjective optimization. Following are m3explorer commands for database handling.

Shell Command name:	**db_change_current**
Arguments:	Name of the db to be set as current - *string*
Options:	-
Description:	The current db is the one actually used for storing exploration results and it is the target of the commands of the shell. All operations performed in m3explorer which act on database has the current db as target. db_change_current allow to define which database have to be used as current, taking its name as parameter. In the case no database with the specified name exists in the m3eplorer environment, such database is automatically generated and set as current.
Example usage:	`> db_change_current "MY_EXPLORATION_DB"`

Shell Command name:	**db_export**
Arguments:	File name into which the current database should be exported - *string*
Options:	-
Description:	This command can be used to export the current database in CSV format to the file specified as parameter.
Example usage:	`> db_export "MY_EXPLORATION_DB.csv"`

Shell Command name:	**db_write**
Arguments:	File name into which the current database should be written - *string*
Options:	-
Description:	Writes the current db on the disk to a specified file in a format readable from m3explorer.
Example usage:	`> db_write "MY_EXPLORATION_DB.db"`

Shell Command name:	**db_report**
Arguments:	-
Options:	-
Description:	Reports all the architectural configurations stored into the current db.
Example usage:	`> db_report`

Shell Command name:	**db_filter_pareto**
Arguments:	-
Options:	-
Description:	Filter the current database keeping only the Pareto points. The Pareto concept is defined once a special variable with the name objectives is declared in the environment as a list of objective metrics. Commands in the example filter the database my_exploration keeping only design points such as no other point is better in terms of energy and delay.

Example usage:	`> db_change_current "my_exploration"`
	`> set objectives={"energy" "delay"}`
	`> db_filter_pareto`

Shell Command name:	**db_plot_objectives**
Arguments:	Databases to be plotted
Options:	-
Description:	Graphical investigation of solutions quality. Plotting functionalities are available only in 2D, thus to use `db_plot_objectives` a special variable with the name objectives must be declared as a list of two objective metrics. If no parameters are passed, the current database only is investigated and objective metrics of stored points are plotted. If one database name parameter is passed, the graphical investigation is performed over such db. In the case two database name parameters are passed, graphical investigation is performed on the same plot for the two databases allowing graphical comparison of solution qualities.

Example usage:	`> set_objectives={"energy" "delay"}`
	`> db_change_current "first_exploration"`
	`> db_read "first_exploration.db"`
	`> db_change_current "second_exploration"`
	`> db_read "second_exploration.db"`
	
	`> ## plot objectives in the second database loaded (the current db)`
	`> db_plot_objectives`
	
	`> ## plot objectives in the first database loaded`
	`> db_plot_objectives "first_exploration"`
	
	`> ## plot objectives in the from both databases`
	`> db_plot_objectives "first_exploration" "second_exploration"`

Shell Command name:	**db_show_optimum**
Arguments:	-
Options:	-
Description:	Displays the optimal point stored in the current db. The optimum point is considered the one with minimum product of objective metrics. Usage example, refers to use case example provided with the distribution of m3explorer.

Example usage:	`> db_read "full_mpeg4.db"`
	`> set_objectives={"energy" "cycles"}`
	`> db_show_optimum`
	
	`Information: Computing optimum point; please note that the cost function is the product of all the metrics`
	`Optimum point: [ ics=16384 dcs=8192 l2cs=131072 icw=1 dcw=8 l2cw=2 iwidth=2 cbs=16 pn=8 ] cost:2.92596e+07`

Shell Command name:	**db_compute_ADRS**
Arguments:	-
Options:	-
Description:	Computes the Average Distance from Reference Set (ADRS). Such distance is a quantitative measure of solution qualities of a multiobjective exploration that is available for whatever dimension of the objective space, thus it can be used even when graphical investigation towards `db_plot_objectives` cannot be performed. ADRS computation needs a reference set that is provided to the command by passing a db name parameters where the reference set is stored. ADRS result is stored in a variable called `last_ADRS`.

Example usage:	`> set objectives="energy" "cycles"`
	
	`> ## performing a full search exploration`
	`> doe_define_doe "m3_full_doe"`
	`> db_change_current "full_DB"`
	`> opt_define_optimizer "m3_pareto_doe"`
	`> opt_tune`
	
	`> ## performing a APRS exploration`
	`> db_change_current "aprs_DB"`
	`> doe_define_doe "m3_random_doe"`
	`> opt_define_optimizer "m3_aprs"`
	`> opt_tune`
	
	`> ## computing aprs quality`
	`> db_compute_adrs "full_DB"`

9 Authors

- Vittorio Zaccaria, *Politecnico di Milano*
- Gianluca Palermo, *Politecnico di Milano*
- Giovanni Mariani, *ALaRI - Università della Svizzera italiana*

10 Acknowledgments

Much of what M3Explorer is today was also defined by the users of the tool. We would like to acknowledge the contributions of the following people, for their early adoption of the tool, their feedback on the tool and on the interfaces, their contributions to the tool and their comments on the manual.

- Cristina Silvano, *Politecnico di Milano*
- William Fornaciari, *Politecnico di Milano*
- Alessandro Sivieri, *Politecnico di Milano*
- Al-Hissi Mohammad, *ALaRI - Università della Svizzera italiana*
- Carlos Kavka, *ESTECO*
- Sara Bocchio, *STMicroelectonics*
- Hector Posadas, *University of Cantabria*

This work is supported by the EC under grant FP7-216693 MULTICUBE (http://www.multicube.eu). The Multicube Explorer tool and the documentation can be found at the following address: http://home.dei.polimi.it/zaccaria/multicube_explorer.