An Automatic Design Space Exploration Framework for Multicore Architecture Optimizations

Horia Calborean, Lucian Vinţan
Advanced Computer Architecture & Processing Systems Research Lab
"Lucian Blaga" University of Sibiu,
Romania
{horia.calborean,lucian.vintan@ulbsibiu.ro}

***Abstract*—- During the last years, especially due to the computing systems complexity growth, the need for tools which perform automatic design space exploration becomes more and more stringent. This paper presents a new initiated project having as the main aim developing a software tool, called FADSE (Framework for Automatic Design Space Exploration), that comes to meet this need. It is intended to provide out-of-the-box algorithms capable of solving single and multiobjective optimization problems. It focuses on automatic design space exploration for multicore and manycore systems. This tool is intended to be flexible, to provide easy development and portability.**

***Index Terms*—Automatic design space exploration, software framework, multicore and manycore architectures**

I. Introduction

In the recent years the multicore architectures became more popular as they improve the performance/power ratio over the single core processor. The number of cores integrated in the processor, not necessarily identical, has risen to tens, hundreds or even thousands (GPUs). These new architectures bring new research challenges [1]. Two of these challenges are discussed in this article: simulation and design space exploration for multicore architectures.

As the number of cores in the processor becomes higher, more configurations have to be simulated and thus the simulation time increases. The cores are communicating through complex interconnection networks having high bandwidth and low latency [2]. Fault tolerance algorithms are implemented in these networks increasing the simulator complexity. In addition the architecture has to be optimized in parallel with the target applications (hardware-software co-design). As the number of possibilities that have to be considered increases more processor configurations have to be evaluated. This leads to an extremely huge search space. The current processor design methodology will not scale and new methods are needed.

Two problems can be identified: increased simulation time and the vast search space.

Reducing simulation time can be done using different techniques [3]: statistical simulation, sampling simulation, parallel simulation.

Even if the simulation time is reduced, investigating all the possible solutions is a NP-hard problem. Exhaustive detailed simulation will not be possible anymore and smarter ways of doing simulation have to be used. Heuristic algorithms are needed to reduce the search space. Techniques borrowed from machine learning and data mining could be used. These methods are needed to search in the ever-increasing space of application, compiler and architecture parameters. The objective is to find the system parameters that lead to the optimum cost/performance ratio for a given real life set of applications.

The remainder of the article is structured as follows: in Section II we present the related work. In Section III the basic concepts of design space exploration are outlined. The developed framework is described in Section IV. Section V shows some preliminary results. Finally, Section VI briefly presents some conclusions and further work.

II. Related Work

Magellan [4] is a complex framework for accelerating multi-core design space exploration and optimization. The algorithms used are: steepest ascent hill climbing, genetic algorithm and ant colony optimization. The framework is bounded to the SMTSIM simulator [5] which is able to run multiprogrammed workloads but not parallel programs. Magellan can be used to find the processor with the highest performance for a given area and power.

ArchExplorer [6] is another automatic design space exploration (ADSE) tool. Researchers can upload their own hardware simulators for different architecture components in order to assess their performance against other designs. The uploaded simulator has to be made compatible with the interface provided by the ArchExplorer developers. After the simulator has been uploaded it automatically becomes part of the design space exploration tool and it is simulated and compared with other similar microarchitectures. The simulator continuously runs on the ArchExplorer servers. Any user can check out any time what is the best configuration found so far for its simulated microarchitecture and also compare it with other implementations.

Another project is *cTuning* (http://ctuning.org/) which focuses more on compilation problems. It is collecting optimization cases from the community to learn how to correlate program features, program and system behavior and good optimizations between multiple programs, datasets, compilers, operating systems and architectures.

M3Explorer [7] is part of a FP7 EC project which aims to develop an automatic design space exploration tool able to easily connect to any existing simulator. It allows optimization of a parameterized system architecture. It is currently integrated with the SESC simulator [8] and it can optimize energy, delay and area or any other objective. M3Explorer integrates several well known search algorithms like: simulated annealing, genetic algorithms, particle swarm optimization, etc. Also it provides an open XML interface for supporting new platforms/architectures.

FADSE is intended to be an easily extensible framework for design space exploration (DSE). Many of the existing tools are bounded to a certain simulator or require the user to write the simulator compatible with the tool. With FADSE we try to make the framework compatible with as many simulators as possible.

Another great advantage of FADSE is the portability provided by JAVA which will enable its use on most of the existing platforms (Windows, Linux and Mac). FADSE can be used as a platform to compare different design space exploration algorithms, too.

III. DSE Basic Concepts

A. Multiobjective optimization problem

Multiobjective optimization is the process of simultaneously optimizing two or more (usually) conflicting objectives. Not only in computer architecture, but in many of the real world problems where optimizations have to be carried out, usually there is more than just one objective. Generally a trade-off between the objectives is necessary. In most cases where multi-optimizations have to be performed there is not a single solution that simultaneously minimizes/maximizes each objective. Multiobjective optimization algorithms are assuring that each objective has been optimized to the extent that if we try to optimize it further, then the other objective(s) will suffer as a result. These solutions are called Pareto solutions. Finding such solutions is the goal when setting up and solving a multiobjective optimization problem.

A general multiobjective problem can be mathematically described as a vector function f where:

$min/max\ \vec{y} = \vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), ..., f_m(\vec{x})]$

subject to:

$\vec{x} = x_1, x_2, ..., x_n \in X$

$\vec{y} = y_1, y_2, ..., y_m \in Y$

where $\vec{x}$ is called the decision vector, X is the parameter space, $\vec{y}$ is the objective vector and Y is the objective space $y_k = f_k(X)$ where $k = 1, 2, 3, ..., m$

B. Pareto optimality

Before describing the concept of Pareto optimality a few definitions have to be introduced [9]:

Definition 1: given two vectors $\vec{x}$, $\vec{y} \in R^n$ we say that $\vec{x} \leq \vec{y}$ if $x_i \leq y_i$ for any $i = 1, ..., n$ and that $\vec{x}$ **dominates** $\vec{y}$ (written as $\vec{x} \prec \vec{y}$) if $\vec{x} \leq \vec{y}$ and $\vec{x} \neq \vec{y}$.

Definition 2: we say that a vector of decision variables $\vec{x} \in X \subset R^n$ is **nondominated** with respect to X, if it does not exist another $\vec{x'} \in X$ such that $\vec{f}(\vec{x'}) \prec \vec{f}(\vec{x})$, where $\vec{f}$ is the multiobjective function.

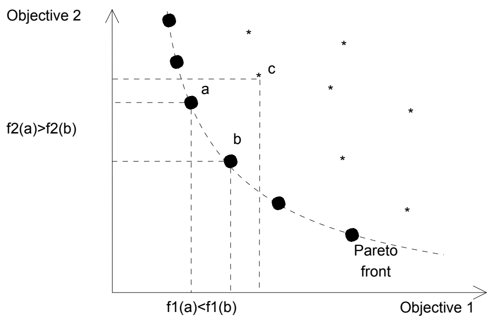

Fig. 1. Pareto front. There is no order that can be established between points a and b. Both a and b are better than c.

Definition 3: a vector of decision variables $\vec{x^*} \in F \subset R^n$ (F is the feasible region) is **Pareto-optimal** if it is nondominatet with respect to F.

Definition 4: the Pareto Optimal Set P^* is defined by:

$$P^* = \{\vec{x} \in F \mid \vec{x} \text{ is Pareto} - \text{optimal}\}$$

Definition 5: the Pareto Front PF^* (see Figure 1) is defined by:

$$PF^* = \{\vec{f}(\vec{x}) \in R^n \mid \vec{x} \in P^*\}$$

The concept of Pareto optimality is very often used in multiobjective algorithms.

IV. The Developed Framework

A. Application structure

The framework is designed to be flexible so that new algorithms, simulators and evaluation metrics can be added easily.

FADSE boots by reading an XML configuration file (see Fig. 2). This XML is a modified version of the XML format used in M3Explorer [7]. Through this XML the user can specify the simulator path (if it is a simulator on which we are running the DSE algorithm; otherwise a synthetic problem can be selected), the parameters and their possible values and the objectives. Beside this, a set of rules (constrains) can be defined. These rules are needed to avoid some impossible/trivial configurations and also to reduce the search space (e.g. it is not worth simulating a configuration where the level 1 cache is larger than the level 2 cache).

The data extracted from the XML file is passed to the Algorithm Runner which configures the framework and starts the design space exploration process. The jMetal library [10] through its integrated algorithms generates individuals which represent simulation configurations. The individuals are passed to the Simulator Wrapper. The Simulator Wrapper converts the data received form the jMetal library, checks if the individual is valid (it obeys the rules specified in the XML file) and passes the simulation configuration to the Simulator Connector.

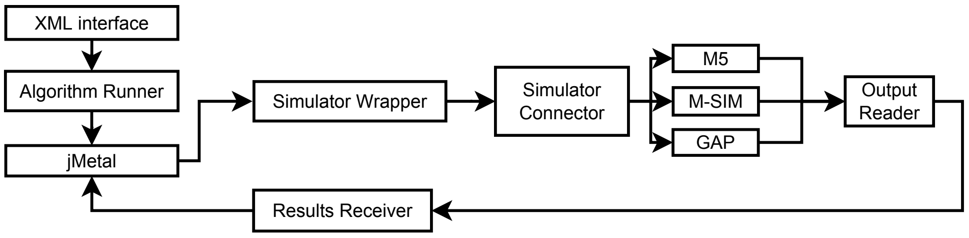

Fig. 2. FADSE current architecture

The Simulator Connector runs the simulator. A Simulator Connector has to be implemented for each simulator since each one has a different interface. After the simulation is done the results are parsed by the Output Reader. The Output Reader has to be implemented for each simulator. We do provide some utility classes to ease the implementation: functions to search for certain strings in files, functions to run native applications and others. The Output Reader sends back the results in a standardized format. The results are passed to the jMetal library by the Results Receiver.

Up to this moment FADSE is able to run and interpret the results of the following simulators GAP [11], M5 (http://www.m5sim.org) and M-SIM (http://www.cs.binghamton.edu/m̃sim/).

Most of the algorithms used for automatic design space exploration are evolutionary algorithms. The currently implemented algorithms are based on Pareto dominance but other algorithms - that solve the multiobjective problem differently - will be implemented: aggregation approaches (they combine the multiple objectives into a single one and reduce the problem to a single objective one), lexicographic ordering (the objectives are ordered and the first objective is optimized then the next one and so on) and sub-population approaches (the population is divided into several sub populations and each try to optimize one objective and at certain points exchange information).

The implemented DSE algorithms can be influenced in the framework by changing the crossover operator and/or the operator that performs the mutation (if the algorithm uses these operators). Also it can change the maximum number of generations that the algorithm should run.

The framework is developed in JAVA which has the advantage of portability. Most of the automatic design space tools implemented are developed for UNIX based system thus reducing the number of simulators that can be used, since simulators have to run on these systems. This inconvenience is eliminated by us through the use of the JAVA platform. Therefore simulators that were designed to run in Windows (like the GAP [11] simulator) and UNIX based systems (M-SIM, M5) can be integrated in the FADSE tool. The speed loss caused by the Java Virtual Machine is not a significant problem since most of the time is spent doing simulations.

B. The parameters

The supported parameters are a subset of the M3Explorer [7] parameters. An example of a certain parameter's format is the following:

```
<parameter name="x" description=""
type="integer" min="0" max="10" step="2"/>
```

In the above example the framework will give values to this parameter from 0 to 10 with a step of 2. The *step* attribute is optional and its default value is 1.

Other types of parameters can be used: the *exp2* type which will produce a geometric series with the ratio equal with 2. The *string* type allows the user to specify a list of string values (e.g. types of cache coherence protocols).

C. The rules

There are three types of rules that can be specified in the input XML. They are the same set of rules found in M3Explorer [7]. We have implemented these rules as it is shown below.

Relational rule:

```
<rule name="minimum cache size" >
 <greater-equal>
  <parameter name="l2CacheSize"/>
  <constant value="2048"/>
 </greater-equal>
</rule>
```

Where the relation can be <greater>, <greater-equal>, <less>, <less-equal>, <equal>, <not-equal>. The *parameter* is a parameter that was defined in the XML The second argument of the rule can be a parameter or a constant value.

And rule:

```
<rule name="cache size check">
 <and>
  <greater-equal>
   <parameter name="l2CacheSize"/>
   <parameter name="l1CacheSize"/>
  </greater-equal>
  <greater-equal>
   <parameter name="l2CacheSize"/>
   <constant value="512"/>
  </greater-equal>
 </and>
</rule>
```

Any number of rules can be inserted in an and rule and any type of rules (*relational rules*, *if rules* and also *and rules*).

If rule:

```
<rule name="cache assoc limitation">
 <if>
  <greater-equal>
   <parameter name="="l2CacheAssoc"/>
   <constant value="1"/>
  </greater-equal>
  <then>
   <equal>
    <parameter name="l1CacheAssoc"/>
    <constant value="1"/>
   </equal>
  </then>
 </if>
</rule>
```

Each individual is validated against these rules before sending it to the simulator. During our experiments these rules covered all the situations we have encountered. If new types of constrains are met, new types of rules can be easily introduced in the framework by extending the *Rule* interface.

D. Implemented algorithms

The SEMO algorithm [12] is one of the simplest genetic algorithms for multiobjective optimization. It stores an archive of all the nondominated individuals. This archive represents the current population. From this population a parent is randomly chosen and mutated by randomly changing a bit. The new individual is accepted if it is not dominated by other individuals and there is no other individual that has the same values for the objectives. All the individuals that are dominated by the new individual are removed from the archive. Then the process is repeated.

The FEMO algorithm [12] is a development of SEMO.

The SEMO algorithm searches for a Pareto optimal solution by mutating a randomly chosen solution from the currently found ones. The problem is that some individuals are chosen more than others to become parents. Because of the uniform sampling technique used to choose the parents, some individuals will get mutated more often just because they were added to the population earlier in history. The neighborhood of these individuals will get over explored. The FEMO algorithm tries to avoid this situation by counting the number of offspring each individual produces. It selects as a parent the individual with the smallest number of offspring. If there are more than one individual with the same number of offspring then the parent is chosen randomly between these individuals.

In SEMO and FEMO if the offspring is not valid a new offspring is produced until all the rules are passed. In the algorithms implemented in jMetal infeasible solutions are included in the offspring population. To reduce de number of individuals the notion of constraint domination [13] is used. The constraint domination works as follows: when individuals are compared if both are feasible the domination relation is used. If one is infeasible the feasible one is selected. If both are infeasible the one that obeys the most rules is kept. If the Simulator Wrapper receives an infeasible configuration it fills the response with default objective values and the simulation is not performed.

Through the jMetal integration more algorithms are available: SPEA2 [18], NSGA-II [13], etc.

The DSE algorithms implemented could be run in parallel in separate threads thus reducing the simulation time when multiple algorithms are compared.

E. Implemented test functions

LOTZ (Leading Ones Trailing Zeroes) [12] is a generalization to two dimensions of the ”Leading Ones” problem. The function $LOTZ : \{0,1\}^n \rightarrow N^2$ problem is defined as:

$$LOTZ(x_1, ..., x_n) = (\sum_{i=1}^{n}\prod_{j=1}^{i} x_j, \sum_{i=1}^{n}\prod_{j=i}^{n}(1-x_j))$$

The objective is to maximize the number of leading ones and the number of trailing zeroes in a bit-string simultaneously. Since there is no point that maximizes both objectives simultaneously the multi objective algorithm will need to find all the nondominated points.

Other implemented test functions include the DTLZ function family introduced by Deb et al. [14]. DTLZ1 was used in this article. The Pareto optimal solutions of DTLZ1 correspond to the hyperplane that intersects the coordinate axes in the objective space at the value 0.5.

F. Implemented metrics

1) Coverage of two sets: The Coverage of two sets metric is used in [15] [16]. Let $X', X'' \subseteq X$ be two sets of decision vectors. The function C maps the ordered pair (X', X'') to the interval $[0,1]$:

$$C(X', X'') = \frac{|\{a'' \in X''; \exists a' \in X' : a' \succeq a''\}|}{|X''|}$$

where$|X''|$ is the cardinal of the set X''.

If all the points from X'' are dominated (or equal) by points from X' then $C(X', X'') = 1$. If $C(X', X'') = 0$ then none of the points in X'' are dominated by points from X'. It should be noted that both $C(X', X'')$ and $C(X'', X')$ should be computed since they would give different values.

It is not required to know the Pareto front to compute this metric.

2) The Error Ratio: The Error Ratio [17] measures the number of individuals in the Pareto optimal set that are not members of the Pareto front. Mathematically:

$$ER = \frac{\sum_{i=1}^{|PF_{known}|} e_i}{|PF_{known}|}$$

where $|PF_{known}|$ is the number of points in the Pareto optimal set. $e_i = 0$ when the i^{th} individual from the Pareto optimal set is an element of the Pareto front (or is not dominated by any point from the Pareto front). $e_i = 1$ if the

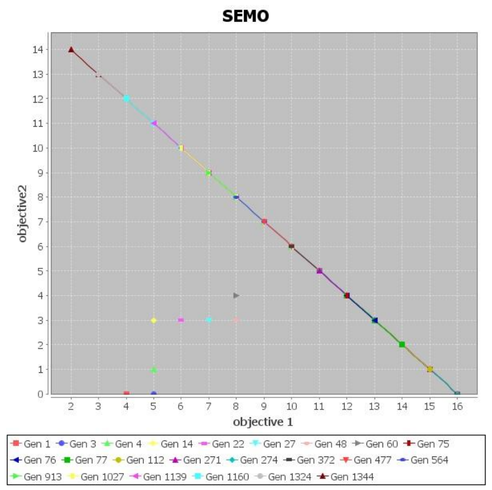

Fig. 3. SEMO algorithm ran on the LOTZ problem with 16 input parameters. After 1500 generations it was not able to determine the entire Pareto front.

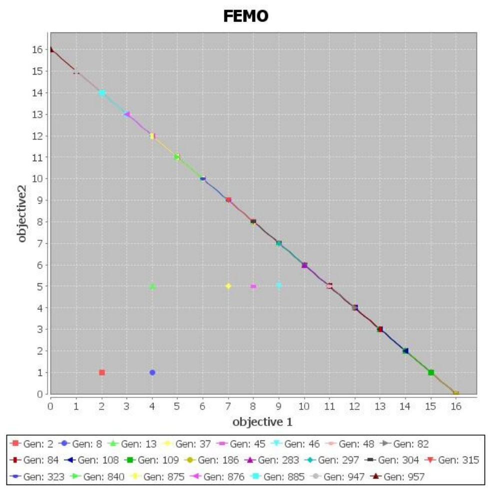

Fig. 4. FEMO algorithm ran on the LOTZ problem with 16 input parameters. After 957 (number of generation not depicted in the figure) generations has found the entire Pareto front

individual from the Pareto optimal set is **not** an element of the Pareto front. If ER is 0 then the solutions from the Pareto optimal set belong to the Pareto front. If ER is 1 then none of the elements from the Pareto optimal set belong to the Pareto front.

The Error Ratio will not provide any useful information if the algorithm does not reach the Pareto front even if the solutions obtained are very close to the Pareto front. This metric can not be used to compare the algorithms on real problems (simulators) because the Pareto front is unknown in this situation.

There are similar metrics that measure the percentage of solutions that belong to the Pareto front (or are not dominated by any of the points from the Pareto front). At the moment of writing this article only LOTZ and DTLZ1 implementations are compatible with the Error Ratio metric.

V. Simulation results

Since both SEMO and FEMO proved to be very sensitive to the choosing of the first individual (which is done random) all the simulations results presented below were run for 1000 times and average values are presented.

SEMO and FEMO algorithms were run on two test problems (LOTZ and DTLZ1). Since the Pareto front is known and finite for the LOTZ problem both algorithms were run until they both discovered all the Pareto optimal solutions. SEMO discovers the entire Pareto front in an average (over 1000 runs) of 1453 generations. For FEMO the average number of generations was only 756. In Figure 3 and 4 the chart generated after a single run is presented. In this certain simulation context SEMO is not able to discover the entire Pareto front in 1500 generations (which was set as the stop condition for the algorithm). Since the Pareto front of the DLTZ1 problem is a hyper plane [14] the number of Pareto optimal solutions is theoretically infinite so the above test was not performed on the DLTZ1 problem.

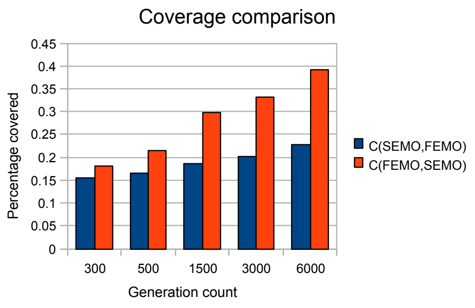

Fig. 5. Coverage comparisons between SEMO and FEMO on the DTLZ1 problem.

The algorithms were run on the DTLZ1 problem for different number of generations (see Figure 5) and the mutual coverages were measured. DTLZ1 was configured to run with seven decision variables and three objectives. As it can be seen in Figure 5 FEMO performs better and more individuals produced by FEMO dominate individuals produced by SEMO as the generation count increases.

Error ratio was not measured on these two algorithms because for the LOTZ problem it will always be 0 if the algorithm reaches the Pareto front since all the other solutions will be dominated by this one solution. The error ratio will be 1 if the Pareto front has not been reached.

Both SEMO and FEMO do not perform well on the DTLZ1 problem since even after 12000 generations the Pareto front has not been reached. This happens because the number of individuals simulated was too small. SEMO and FEMO generate only one individual in each generation. It is reported in [14] that NSGA-II [13] reaches the Pareto front in 300 generations with a population of 100 so the number of evaluated individuals is larger even with this more advanced algorithm.

VI. Conclusions and further work

The implemented algorithms are able to solve the LOTZ problem with a fairly small amount of simulated individuals. An exhaustive simulation would have needed 65536 (2^{16}) evaluations for 16 decision variables. SEMO needed in average 1453 evaluations (around 2% of the whole effort) and FEMO only 756 evaluations in average (around 1% of the effort). These reductions represent great improvements and demonstrate the power of these heuristic algorithms.

Both SEMO and FEMO have proven not to be able to reach the Pareto front in a reasonable number of simulations on more complex problems like the DTLZ problems family. More advanced algorithms are needed. SEMO and FEMO provide a good example on how to implement a multiobjective algorithm in FADSE.

The next step will be to fully integrate the jMetal library in FADSE so that we can use the implemented multiobjective algorithms (like SPEA2 [18], NSGA-II [13]).

Another development of the framework will be to integrate a network module. This will make it possible to run simulations in parallel on different machines and collect the data to a central server. After implementing the network module the focus will be in parallelizing the simulations. Many search algorithms provide a huge amount of intrinsic parallelism. Being able to run the algorithm in a distributed manner will greatly reduce the time needed for simulations. Most of the algorithms found in the literature (like SPEA2 and NSGA-II) have points of synchronization (fitness assignment). The simulation can not continue until all the other simulations finish. Improved algorithms are needed that can pass this constrain and also provide good results.

Connectors will be written to allow the optimization of more simulators. The focus will be primarily on GEMS and NS3 (http://www.nsnam.org/). More metrics will be added to the framework, too.

The framework will integrate a database system which will store the simulation results. If the same configuration has to be simulated again (even if different algorithms require it) the results could be reused from the database.

The network module, the algorithm parallelization and the database integration will allow us to perform automatic design space exploration on real simulators.

After more algorithms will be implemented a comparison between them will be performed on synthetic problems (like DTLZ family of problems) and especially on existing processor simulators. This will help us find the best suited algorithm for a certain problem.

VII. Acknowledgements

This work was partially supported by CNCSIS no. 485/2008 research grant offered by the Romanian National Council for Academic Research. Also we would like to thank Prof. Dr. Theo Ungerer and to his research group for their advices and help during the first author's PhD stage at Augsburg University (2009/2010, 5 months period).

References

[1] L. Vintan, "Directii de cercetare in domeniul sistemelor multicore / main challenges in multicore architecture research," *Revista Romana de Informatica si Automatica*, vol. 19, no. 3, 2009.

[2] E. Salminen, A. Kulmala, and T. D. Hamalainen, "Survey of network-on-chip proposals," *white paper, OCP-IP*, 2008.

[3] J. J. Yi and D. J. Lilja, "Simulation of computer architectures: Simulators, benchmarks, methodologies, and recommendations," *IEEE Transactions on Computers*, vol. 55, no. 3, pp. 268–280, 2006.

[4] S. Kang and R. Kumar, "Magellan: A framework for fast multi-core design space exploration and optimization using search and machine learning," *Design, Automation and Test in Europe*, vol. 51, pp. 1432–1437, 2008.

[5] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," in *25 years of the international symposia on Computer architecture (selected papers)*. Barcelona, Spain: ACM, 1998, pp. 533–544.

[6] V. Desmet, S. Girbal, O. Temam, and B. France, "Archexplorer. org: Joint compiler/hardware exploration for fair comparison of architectures," in *INTERACT workshop at HPCA*, 2009.

[7] M. O. OF and M. P. SOC, "FP7-216693-MULTICUBE project." [Online]. Available: http://www.multicube.eu/

[8] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, *SESC simulator*, 2005. [Online]. Available: http://sesc.sourceforge.net

[9] M. Reyes-Sierra and C. A. Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *International Journal of Computational Intelligence Research*, vol. 2, no. 3, pp. 287–308, 2006.

[10] J. J. Durillo, A. J. Nebro, F. Luna, B. Dorronsoro, and E. Alba, "jMetal: a java framework for developing Multi-Objective optimization metaheuristics," Departamento de Lenguajes y Ciencias de la Computacion, University of Malaga, E.T.S.I. Informatica, Campus de Teatinos, Tech. Rep. ITI-2006-10, Dec. 2006.

[11] S. Uhrig, B. Shehan, R. Jahr, and T. Ungerer, "A two-dimensional superscalar processor architecture," in *The First International Conference on Future Computational Technologies and Applications, Athens, Greece*, 2009.

[12] M. Laumanns, L. Thiele, E. Zitzler, E. Welzl, and K. Deb, "Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem," in *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*. Springer-Verlag, 2002, pp. 44–53.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[14] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *E-Commerce Technology, IEEE International Conference on*, vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, 2002, pp. 825–830.

[15] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," *IEEE transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[16] G. Palermo, C. Silvano, and V. Zaccaria, "Discrete particle swarm optimization for multi-objective design space exploration," in *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*, 2008, pp. 641–644.

[17] C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 1st ed. Springer, Jun. 2002.

[18] E. Zitzler, M. Laumanns, L. Thiele *et al.*, "SPEA2: Improving the strength Pareto evolutionary algorithm," in *EUROGEN*, 2001, pp. 95–100.