"Lucian Blaga" University of Sibiu
"Hermann Oberth" Engineering Faculty
Computer Engineering Department

Multi-Objective Optimization of Advanced Computer Architectures using Domain-Knowledge

PhD Thesis

Author:
Horia Andrei Calborean, M.Sc.

PhD Supervisor:
Professor Lucian Vinţan, PhD

SIBIU, September 2011

"Lucian Blaga" University of Sibiu
"Hermann Oberth" Engineering Faculty
Computer Science Department

Multi-Objective Optimization of Advanced Computer Architectures using Domain-Knowledge

PhD Thesis

Author:
Horia Andrei Calborean, M.Sc.

PhD Supervisor:
Professor Lucian Vinţan, PhD

SIBIU, September 2011

Acknowledgements

First, I would like to express my gratitude to Professor *Lucian Vinţan*, who is the coordinator of this PhD Thesis. I appreciate the trust he placed in me by offering me the opportunity to work on this research project. A sincere thank you to him for being there throughout this adventure and helping me successfully complete this Thesis. I truly appreciate his useful comments and guidance.

I would also like to thank Professor *Theo Ungerer* from the University of Augsburg for the training period I spent with his research group and also for the successful collaboration that followed. I want to especially thank *Ralf Jahr*, with whom I have cooperated and who has become a close friend to me.

I want to express my gratitude to my friend and colleague *Ciprian Radu* for his useful advice and for supporting me throughout this period.

Special thanks to the teachers from the Computer Science Department of "Lucian Blaga" University of Sibiu, especially to Associate Professor Dr. Ing. *Remus Brad* who reviewed part of my work during this PhD period.

I would also like to thank Associate Professor Dr. Ing. *Adrian Florea* and Assistant Professor Dr. Ing. *Árpád Gellért* for their support, constructive comments and the good collaboration we had together throughout the years, even before I started my PhD.

Many thanks to the team lead by Professor *Nicolae Ţăpuş* for granting me access to the HPC system from Politehinca University from Bucharest. I want to mention Associate Professor Dr. Ing. *Emil Slusanschi* and Assistant Professor *Alexandru Herişanu* for the help and assistance they offered.

For this research I have collaborated with very good students from "Lucian Blaga" University of Sibiu: *Andrei Zorila, Camil Banciou* and *Radu Chis*. I would like to thank them for their help.

I would also like to thank to some people very dear to me: my wife, *Angela*, and my parents for being there for me when I needed them.

This work was supported by POSDRU financing contract POSDRU 7706.

Contents

Author's Papers

Published Papers

Ralf Jahr, Theo Ungerer, **Horia Calborean**, Lucian Vinţan "*Automatic Multi-Objective Optimization of Parameters for Hardware and Code Optimizations*", The 2011 International Conference on High Performance Computing & Simulation(HPCS 2011), 4 – 8 July, 2011, Istanbul, Turkey. Selected for **Outstanding paper award**. Received **special invitation** to publish an extended version in journal: "**Concurrency and Computation: Practice and Experience**" Wiley – impact factor 0.907. Indexed IEEE

Horia Calborean, Lucian Vinţan "*Framework for Automatic Design Space Exploration of Computer Systems*", Acta Universitatis Cibiniensis – Technical Series, "Lucian Blaga" University of Sibiu, Romania, ISSN 1583-7149, May 2011, Sibiu, Romania

Horia Calborean, Ralf Jahr, Theo Ungerer, Lucian Vinţan "*Optimizing a Superscalar System using Multi-objective Design Space Exploration*", 18th International Conference on Control Systems and Computer Science (CSCS 18), IEEE Romanian Chapter, 24 - 27 May, 2011, Bucharest, Romania. **Selected to be published by Elsevier**.

Horia Calborean, Lucian Vinţan "*Toward an efficient automatic design space exploration frame for multicore optimization*", Sixth International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES), July 2010, Terrassa (Barcelona), Spain.

Horia Calborean, Lucian Vinţan "*An automatic design space exploration framework for multicore architecture optimizations*", In Proceedings of the 9-th IEEE RoEduNet International Conference, Sibiu, Romania, June 2010. **Best paper award**. Indexed ISI Thomson Reuters Proceedings, IEEE, SCOPUS

Ciprian Radu, **Horia Calborean**, Adrian Florea, Árpád Gellért, Lucian Vinţan "*Exploring some multicore research opportunities. A first attempt*", Fifth International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES), Academic Press, Ghent, Belgium, pp. 151-154, ISBN 978 90 382 1467 2, July 2009, Terrassa (Barcelona), Spain.

Adrian Florea, Ciprian Radu, **Horia Calborean**, Adrian Crapciu, Árpád Gellért, Lucian Vinţan "*Understanding and Predicting Unbiased Branches in General-Purpose Applications*", Buletinul Institutului Politehnic Iasi, Tome LIII (LVII), fasc. 1-4, Section IV, Automation Control and Computer Science Section, Zentralblatt MATH indexed, pp. 97-112, ISSN 1220-2169, "Gh. Asachi" Technical University 2007, Iasi, Romania, Indexed Zentralblatt MATH.

Adrian Florea, Ciprian Radu, **Horia Calborean**, Adrian Crapciu, Árpád Gellért, Lucian Vinţan "*Designing an Advanced Simulator for Unbiased Branches' Prediction*",
Proceedings of 9th International Symposium on Automatic Control and Computer Science, ISSN 1843-665X, November 2007, Iasi, Romania.

Ciprian Radu, **Horia Calborean**, Adrian Crapciu, Árpád Gellért, Adrian Florea "*An Interactive Graphical Trace-Driven Simulator for Teaching Branch Prediction in Computer Architecture*", The 6th EUROSIM Congress on Modeling and Simulation, (EUROSIM 2007), ISBN 978-3-901608-32-2, 9-13 September 2007, Ljubljana, Slovenia (special session: Education in Simulation / Simulation in Education I).

Submitted Papers

Ralf Jahr, **Horia Calborean**, Theo Ungerer, Lucian Vinţan, "*Boosting Design Space Explorations with Existing or Automatically Learned Knowledge*," The 16-th International GI/ITG Conference on Measurement, Modeling and Evaluation of Computing Systems and Dependability and Fault Tolerance (Submitted), 2012, Kaiserslautern (Germany)

Árpád Gellért, **Horia Calborean**, Lucian Vinţan, Adrian Florea, "*Multi-Objective Optimizations for a Superscalar Architecture with Selective Value Prediction*," IET Computers & Digital Techniques (submitted, manuscript ID: CDT-2011-0116).

Workshops

FADSE was presented at HiPEAC Computing Systems Week, Chamonix, April 2011 by Ralf Jahr in a presentation called "**FADSE and GAP:** Design Space Exploration for the Grid Alu Processor (GAP) with the Framework for Automatic Design Space Exploration (FADSE)"

Technical Reports

Horia Calborean, "*Developing a framework for ADSE which connects to multicore simulators*," Computer Science Department, "Lucian Blaga" University of Sibiu, 2010, Sibiu, Romania.
Horia Calborean, "*An overview of the multiobjective optimization methods,"* Computer Science Department, "Lucian Blaga" University of Sibiu, 2010, Sibiu, Romania.
Horia Calborean, "*An overview of the features implemented in FADSE," Computer Science Department*, "Lucian Blaga" University of Sibiu, 2011, Sibiu, Romania.
Horia Calborean, "*Introduction to the MANJAC system*," Computer Science Department, "Lucian Blaga" University of Sibiu, 2011, Sibiu, Romania.

"If you don't work on important problems,
it's not likely that you'll do important work."

Richard Hamming

1 Introduction

As technology has advanced, computer systems have become more and more complex [1][2][3][4]. When designing such a system, an architect must take into account many parameters. The design space of a microprocessor-compiler ensemble can reach millions of billions of possible configurations. A microprocessor can not go into production until it is not evaluated to meet the performance criteria. Evaluating a single configuration can take hours or even days. Therefore an exhaustive evaluation is infeasible.

The current approach is to use human experts to select candidate configurations, evaluate them on computer simulators and then try to optimize them. With the growth in complexity and with the increasing number of integrated heterogeneous cores, the task of finding good configurations becomes very hard for a designer.

The problem is further exacerbated by the fact that not only performance needs to be optimized: power consumption, area integration became very important objectives. Finding relations between parameters of the architecture and the way they influence the multiple objectives that need to be optimized proves to be difficult.

One solution to this problem is to use tools that perform automatic design space exploration (DSE) using different heuristic search algorithms. In the HiPEAC vision [5] automatic design space exploration (ADSE) is viewed as one of the most important problems that need to be solved in the following years. Heuristic search algorithms have been used for NP-hard problems for long time. In the recent years, the computer designers have shown an increased interest for them. They are currently the one of the few viable solutions to NP hard problems, like the one of design space exploration.

The **scope** of this PhD thesis is to perform multi-objective optimization of advanced computer architectures using experts' domain-knowledge. For this we have to fulfill the following **objectives**:

- analyze the state of the art heuristic algorithms and classify them;
- determine how they could cope with real life problems, where constraints between the parameters might exist;
- find methods to measure their performance;
- develop a robust and fast DSE framework that can connect to any existing computer simulator;
- include multiple heuristic algorithms into this framework;
- research how domain-knowledge could be easily integrated in this framework and how it could influence the heuristic algorithms;
- evaluate the algorithms on several computer simulators; the simulators should range from single to multi-core and even to system on chip simulators;
- perform comparisons between the algorithms and determine the impact of domain-knowledge on the results.

"It's so much easier to suggest solutions
when you don't know too much about the problem."

Malcolm Forbes

2 Design Space Exploration Algorithms

As we already stated in Chapter 1 we are dealing with NP hard multi-objective search problems. Manual design space exploration is not feasible so new methods are required. One of them is to use multi-objective algorithms to perform this task automatically.

We use heuristic-stochastic search methods that we divided into two classes: evolutionary algorithms and bio-inspired algorithms. All the used algorithms are based on the Pareto efficiency:

2.1 Multi-Objective Search Algorithms

In Figure 2.1-1 the Pareto front for a bi-objective minimization problem can be seen. The points represent individuals. Point *c* is not on the Pareto front because it is dominated by both points, *a* and *b*. Points *a* and *b* are not strictly dominated by any other, and hence do lie on the frontier.

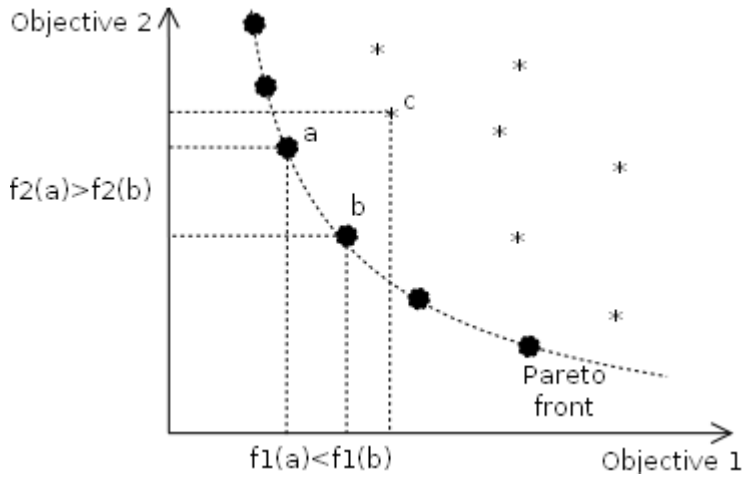

Figure 2.1-1 Pareto front

The problem with the Pareto front is that it does not give us an ordering between the points (for example points *a* and *b* in the figure above).

The following definitions are used in [6]:

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 multi-objective function.

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 nondominated with respect to *F*.

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

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

Definition 5: the Pareto Front PF^* is defined by:

$PF^* = \left\{ \vec{f}(\vec{x}) \in R^n \,|\, \vec{x} \in P^* \right\}$

We call Pareto front approximation (or known Pareto front), the Pareto optimal solutions in the objective space, found by an algorithm up to a point. The

individuals, that form the Pareto front approximation, are called, in the parameter space, the Pareto set.

2.1.1 SEMO and FEMO Experiment

We used in our first experiments two simple evolutionary algorithms: SEMO and FEMO [7]. SEMO stores an archive of all the non-dominated individuals. This archive represents the current population. From this population a parent is chosen randomly and mutated by randomly changing a parameter. The new individual is accepted in the archive if it is not dominated by other individuals and there is no other individual that has the same values for the objectives. If there are individuals in the archive dominated by this new one, they are eliminated. FEMO is an improvement of SEMO. The difference is that: when selecting the parent, the algorithm deterministically chooses the individual with the smallest number of offspring.

We tested these two algorithms on two synthetic test problems LOTZ [7] and DTLZ1 from the DTLZ family of problems [8]. We concluded that both algorithms were able to solve the LOTZ problem in a relatively small amount of time (2% and 1% of the effort required by an exhaustive evaluation for SEMO and FEMO respectively). The DTLZ1 problem was not solved by any of the algorithms (no Pareto optimal points were found). These results were published by us in [9].

2.1.2 NSGA-II

NSGA-II is a genetic algorithm developed by Deb et al. [10]. The algorithm works as follows: it first generates an initial population and evaluates it (this will be the parent population). From this initial/parent population an offspring population is generated using crossover and mutation. The two populations (parent and offspring) are united into a single one and sorted (fitness assignment) according to the domination relationship and a density information (crowding). From this sorted union the best individuals (best fitness) are selected and they will form the new parent population and the process is repeated.

2.1.3 PEA2

SPEA2 introduced by Zitzler et al. [11] is another genetic algorithm. The only differences between NSGA-II and SPEA2 are: (a) SPEA2 uses an external population (archive) to keep nondominated individuals and (b) it assigns fitness in a different manner. First, the strength of each individual is computed (which is equal with the number of individuals the current individual dominates). Then the raw fitness is computed. The raw fitness is the sum of strengths of the individuals that dominate current individual. To this raw fitness, the density information is added, which is the inverse distance to the nearest k^{th} neighbor, to obtain the fitness of the individual. The algorithm then selects the best individuals (from both the archive and the offspring population) and stores them in the archive. The archive is then used as a parent population.

2.1.4 Comparison between NSGA-II and SPEA2

SPEA2 tends to retain many duplicates in its archive during the environmental selection process (see [11]) partially because of the density computation method. In SPEA2 identical individuals will have the same fitness assigned (only one neighbor is used to compute the density), while in NSGA-II these individuals can have a different fitness assigned. This happens because of the crowding assignment [10]: two

individuals are used to compute the crowding, one on each side of the current individual (sorted according to an objective). This means that twins will have one identical individual, but there is a high probability, that the other one is a different individual. In SPEA2 there is a chance that both individuals get selected and passed to the next generation while in NSGA-II one of them might be discarded. During the selection process, all the individuals have the same chance of becoming parents, but since in SPEA's archive there are more duplicates than in NSGA-II, they have a greater chance of being selected and produce the same individuals. Another problem that leads to duplicates is the archive used in SPEA2. SPEA2 stores in the archive only the nondominated individuals. Parents are selected only from the archive but in our explorations we noticed that there are usually fewer nondominated individuals than the maximum size of the archive/population. As a result the archive will be quite empty (hence a greater chance to select the same parents again) while in NSGA-II the population is filled with worse individuals until it is full. The advantage of SPEA2 is that these duplicated individuals do not have to be simulated again due to the integrated database. The disadvantage is that SPEA2 does not have such a good spread of solutions (even if at the end there are 100 individuals many of them are duplicates).

2.2 Multi-objective Particle Swarm Optimization

In this chapter we will present some bio-inspired algorithms. They are based on the way birds search for food.

2.2.1 OMOPSO and SMPSO

OMOPSO [12] is a particle swarm optimization method (bio-inspired). In these algorithms the population is called swarm. The individuals are called particles, which are "flown" through space following the best performing particle at that moment. The position of a particle is given by the current values of its parameters, belonging to an orthogonal representation particle's space. As every particle tries to get closer to the current best particle its parameters are changed. The change takes into account both the current global best and the particle's personal best. Based on this change, the particle gets a new position and it needs to be evaluated again. After all the particles are evaluated, the new global best particle is selected, the personal bests are updated and the process is restarted. In multi-objective PSO algorithms there can be multiple global best particles. The approach when selecting a leader is similar with selecting parents in NSGA-II.

SMPSO [13] is a development of OMOPSO. The most notably change is that it ads a method of constraining the maximum speed a particle can reach.

2.3 Handling Constraints

Most of the real world problems are constrained problems. Methods of handling these constraints are needed. The method we used for constraints handling is proposed in [10] and is employed in couple with binary tournament selection. The solution adopted is:

- if both individuals are feasible then the domination relation is used;
- if one individual is feasible and one is not feasible, the feasible one is selected;
- if both individuals are infeasible then the one that violates the least constrains is selected. If this can not be determined or both individuals violate the same number of constrains the selected individual is chosen randomly.

2.4 Measuring Multi-objective Algorithms Performance

2.4.1 Hypervolume

This metric was used by Zietzler and Thiele [14] and Coelo [15]. Let $X' = (x_1, x_2, ..., x_k) \subseteq X$ be a set of decision vectors (individuals). The function $S(X')$ gives the volume enclosed by the individuals and the axes in the objective space.

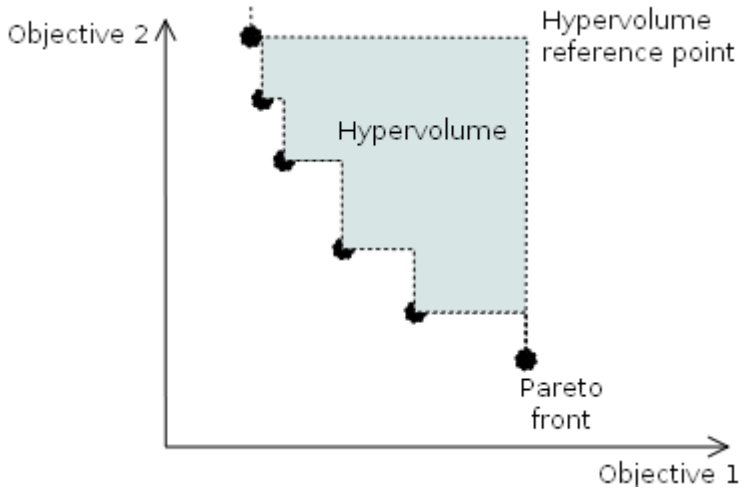

Figure 2.4-1 Hypervolume computation for a minimization problem

When the objectives have to be minimized a point has to be established, called hypervolume reference point, which will replace the origin in the hypervolume computation. The hypervolume reference point coordinates are set to the maximum values of the objectives (see Figure 2.4-1). The hypervolume value represents the percentage covered by the volume enclosed between the hypervolume reference point and the Pareto front approximation, from the total volume enclosed between the hypervolume reference point and the axes (the values are normalized).

2.4.2 Two Set Difference Hypervolume

This metric was proposed by E. Zitzler in his PhD thesis [16]. Two Set Difference Hypervolume (TSDH) is defined as: $TSDH(X', X'') = H(X' + X'') - H(X'')$ where $X', X'' \subseteq X$ are two sets of decision vectors, $H(X)$ is the hypervolume of the space covered by the decision vector *X*, and $X' + X''$ is the nondominated decision vector obtained after the union of X' and X''.

$TSDH(X', X'')$ computes the hypervolume of the space that it is dominated by X' but not by X''.

2.4.3 Coverage of Two Sets

This metric was used by Zietzler and Thiele [14] and Palermo [17]. It computes the percentage of individuals from a populations dominated by individuals from another population.

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]:

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

"Strive not to be a success, but rather to be of value."

Albert Einstein

3 Developing FADSE: A Framework for Automatic Design Space Exploration

Framework for Automatic Design Space Exploration (FADSE) is a tool, developed by us, which is able to perform automatic design space exploration using a large variety of algorithms. FADSE is able to run the evaluations in a distributed manner on Local Area networks (LAN) or High Performance Computing (HPC) systems. It is reliable: can recover from crashed network, crashed clients or power loss. It includes many features that allow the user to intervene and input his/hers knowledge inside the search algorithm. All this information can be provided into a human readable form using a simple interface (some inspired from the M3Explorer tool [18]).

FADSE integrates the jMetal library [19] which provides many state-of-the-art multi-objective search algorithms. We have extended jMetal and we are able to perform design space exploration with simulators like: Multi2Sim [20], GAP [21], M5 (http://www.m5sim.org) and M-SIM (http://www.cs.binghamton.edu/~msim/). We have included multi-objective test functions (LOTZ) and metrics: error ratio, coverage of two sets, hypervolume, hypervolume two set difference.

FADSE is developed in JAVA and can be run on different operating systems.

The latest version of FADSE can be downloaded from http://code.google.com/p/fadse/.

3.1 Accelerating DSE through Distributed Evaluation

The framework has been designed as a client-server application. The server runs the DSE algorithm and the clients perform the simulations.

We changed some of the algorithms implemented in jMetal to work in a distributed manner. Most of the evolutionary algorithms generate an offspring population and only after all the individuals are evaluated they are used. Taking advantage of this behavior the evaluation process can be distributed easily. The offspring individuals are evaluated in parallel by FADSE (see Figure 3.1-1). If multiple benchmarks need to be run to evaluate a single configuration, then these separate evaluations are also done in parallel.

Next FADSE collects all the results for a single individual and computes the average. As an example, for an algorithm with an offspring population of 100, where each individual has to be evaluated on 10 benchmarks, we reach 1000 simulations that can be run in parallel.

FADSE is designed to cope with the failures of clients or the network. If a client does not respond, the task, which was scheduled on it, is sent to another client. After a number of retries the job is marked as infeasible if none of the clients manages to complete it. This allows the exploration to continue even if some configurations are proven to be impossible to simulate. If the whole system has to be restarted FADSE can continue its work because of the built in checkpointing mechanism.

The clients have recovery mechanisms implemented. When a client is started, in parallel a watchdog timer thread is also executed. If the client does not receive messages for a period of time and it is not simulating a configuration, it is automatically restarted. This prevents situations were a client might be blocked on something unexpected.

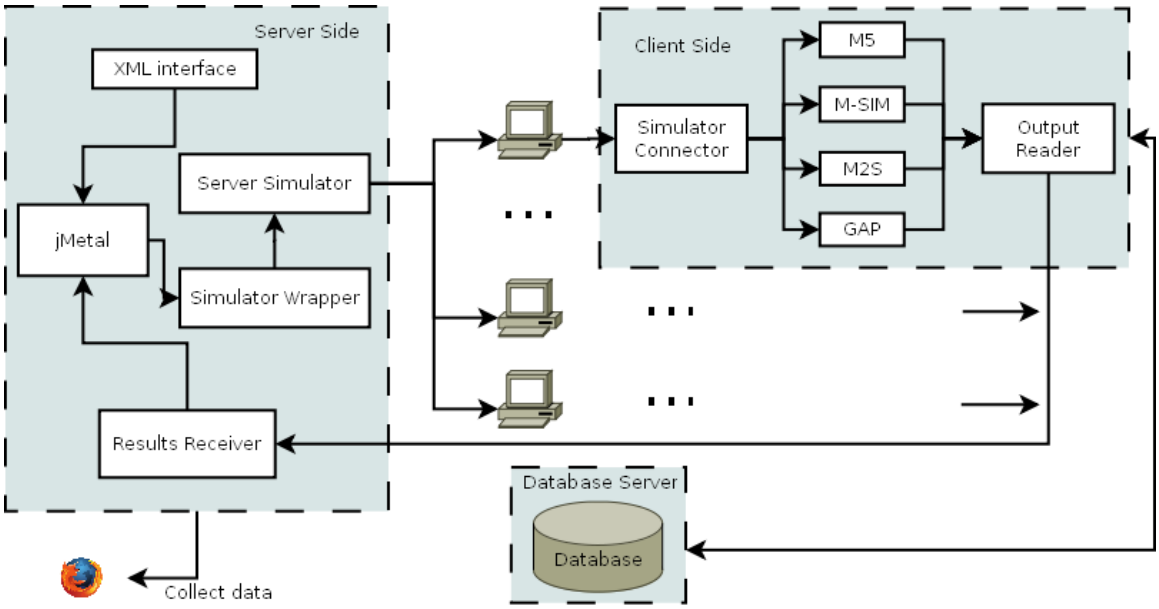

Figure 3.1-1 Structure of FADSE (distributed version)

With this server-client configuration in place, we were able to run on local area networks, HPC systems and multi-processor virtual machines. Some of the systems on which we have tested FADSE are presented below.

As a local area network we have used 9 Intel dual-core machines. They were configured in VPN (Virtual Private Network) to assure that their IP's did not change over time (they were configured using DHCP and power loss was frequent). One computer was chosen as the server. On this computer the DSE algorithm and the MySQL server (see Chapter 3.2 about the reuse scheme implemented in FADSE) were running. On all the other machines, two FADSE clients were started. This system was used to obtain some of the results presented in Chapter 5. The system had a combination of Windows XP and Windows 7 machines.

FADSE was used on two HPC systems: one residing at University "Lucian Blaga" from Sibiu and one in Bucharest from the Politehnica University. The one from Sibiu was extensively used. All the results presented in Chapters 6 and 7 were obtained using this system. All these clusters use Red Hat Linux as an operating system.

The HPC system from Sibiu (http://zamolxe.hpc.ulbsibiu.ro/ganglia/) contains two clusters: one based on Intel Xeon quad core processors (CSAC cluster) and one based on IBM Cell processors (ACAPS cluster).

The CSAC cluster contains 15 blades with two processors and 4GB of DRAM. This means there are 8 cores on each blade, which leads to 120 cores in the whole system. In our experiments we have used one node (head node) as the server and no client was run on this node (this node was used by many other services, including MySQL). On all the other nodes clients were started (8 on each one).

The ACAPS cluster contains two blades, each blade having two Cell processors. Each Cell processor contains a multithreaded PowerPC core and 9 specialized processing units. We have tested successfully FADSE with this system too. We started a server on the head node on the CSAC cluster and the clients on the ACAPS cluster.

Successful tests were conducted on the HPC from Bucharest. This system allows starting processes using batch commands. Some special scripts were required

to map FADSE on this system, but in the future we plan to integrate the possibility to use batch commands from within FADSE using some specialized connectors. Tests were conducted using around 100 clients.

We present one final use case: a Windows 7 based virtual machine with 32 processors from the University of Augsburg. We used this machine to obtain most of the results in Chapter 5. We either ran a single server with 32 clients (the server does not consume much processor time, and it is idle most of the time during simulation), or two servers in parallel with 16 clients each.

Multiple servers can be started on the same machine and they can also run alongside clients. This means that multiple DSE processes can be started in parallel on the same system, so all the resources available can be used.

3.2 Accelerating DSE through Results Reuse

DSE algorithms tend to produce the same individuals again after some generations. Instead of simulating them once more we thought to reuse the results from a database. For this we have connected FADSE with a Database Management System (DBMS).

The integration with the database proved to be very successful. We reached a reuse of around 67% during a 100 generations run with a population of 100 individuals. Also since we ran DSE processes with the same simulator but in different context, results could be used from previous explorations leading to an even greater reuse.

3.3 Universal Interface – Connectors

FADSE is designed in such a manner that it can be connected to almost any existing simulator with a minimal effort and in most situations with no changes to the simulator (source code is not necessary).

3.4 Extensible Input XML Interface

A XML interface is used to configure FADSE, the design space and the specific constrains. First the user has to specify the simulator connector he/she wants to load and the set of parameters required by the connector.

Then a list of benchmarks is specified, the database connection is configured and the list of parameters that need to be varied (these parameters can be of type integer, arithmetical progression, geometrical progression, list of strings, etc.) is specified.

"Information is not knowledge.
The only source of knowledge is experience."

Albert Einstein

4 Improving FADSE with Domain-specific Knowledge

All the algorithms included in FADSE are general ones. They were designed to solve many types of problems. Specialized algorithms, that include knowledge about the problem to be solved, might provide better results, but they can be applied to a single problem.

We still want FADSE to be a general framework, a tool that can be used with many simulators/problems. The goal of this chapter is to identify some methods to express knowledge in an easy manner and to include it into FADSE without loosing generality. This knowledge will be then used by the design space exploration algorithms.

4.1 Design Space Constraints

Constraints are needed when optimizing processor architectures to avoid impossible configurations or configurations that the designer knows will not lead to good results. One of the best examples is that the size of the level 2 cache has to be bigger then the size of the level 1 cache, otherwise it does not make any sense. For a DSE algorithm, the *size* parameter has no meaning so it might generate configurations where the above rule is not respected.

When designing the interface for FADSE, through which the user can specify the constraints, we used as a model the M3Explorer tool. The implementation is however original. Migrating from M3Explorer to FADSE should not be difficult since FADSE's interface is mostly a superset of the M3Explorer interface. The constraints can be easily expressed from within the input XML configuration file.

The constraints implemented in FADSE are one of its most powerful features. They give the user a good control over the size of the design space and its borders. Constraints help the algorithm avoid exploring uninteresting areas, resulting in a faster DSE process. They have been used extensively during our experiments (see Chapter 6).

4.2 Hierarchical Parameters

4.2.1 Motivation

In many designs there are parameters which validate or invalidate another set of parameters. For example the parameter "branch predictor type" will validate or invalidate the parameters associated with a specific value of this parameter. If the branch predictor type is set to a two level adaptive predictor the active parameters might be table sizes, history length and other. If the branch predictor is a neural predictor then another set of parameters will be active. These might lead to problems during an evolutionary algorithm.

To solve this problem we proposed and developed an XML interface that allows the user to specify the hierarchy of parameters and also some new specialized genetic operators for crossover and mutation.

4.2.2 Adapting Genetic Operators

4.2.2.1 Crossover

The crossover operator receives the tree and two individuals. It switches from the current encoding of the individual to a tree encoding. At the same time it determines the valid and invalid edges. Crossover can be performed on edges that do not point to an invalid node or on the root nodes.

4.2.2.2 Mutation

The mutation operator is simpler. The individual to mutate is inserted in the tree and the array of binary values is extracted. One of the values is randomly picked and mutated. Of course this is done only taking into consideration the mutation probability.

4.3 Introducing Domain-specific Knowledge through Fuzzy Logic

In this paragraph we propose to use fuzzy rules for representing domain knowledge in an easy to use language by the designer that can be also understood by our DSE tool. As far as we know, we are the first ones to use fuzzy rules as a method to express a priori knowledge into a design space exploration tool for computer architectures.

These rules allow us to write statements like:

IF level 1 cache size IS small THEN level 2 cache size IS big

4.3.1 Mamdani Rules-system

From all the possible inference systems we have focused on the Mamdani inference system [22]. For this we have used the MIN and MAX functions for AND and OR respectively. For the implication we used the Mamdani implication (MIN). For consequent aggregation we used the MAX function [23][24].

4.3.2 Integrating Fuzzy Logic into FADSE

4.3.2.1 FCL Language

The Fuzzy Control Language (FCL) [25] has been used to describe fuzzy functions. FCL is a standard language for fuzzy control programming and has been published by the International Electrotechnical Commission (IEC) [26]. The language specification can be found in IEC document 61131-7. A draft version can be found at [25]. We integrated the jFuzzyLogic library [27] into FADSE to be able to use the FCL specification and the included inference systems. FADSE accepts as an input a file written in FCL.

4.3.2.2 Mutation Operators

To use the information provided by the fuzzy rules we had to change the genetic operators.

4.3.2.2.1 Changing the Bit-flip Mutation Operator

In this work the bit flip mutation operator was extended to take into consideration the information provided by the output of the fuzzy rules.

The classical bit flip mutation is changed as follows:

1. For all the variables (genes) in the individual (chromosome);
   1.1. If a fuzzy rule exists for this parameter
      1.1.1. Do mutation (with a certain probability) taking into consideration the information provided by fuzzy rules;
   1.2. Otherwise (do bit flip mutation);
      1.2.1. Generate a random number between 0 and 1;
      1.2.2. If the random number is smaller than the probability of mutation;
         1.2.2.1. Change the current variable to a random value;
2. STOP.

The only change to the bit flip mutation algorithm is that: if the current variable is defined as an output variable in the FCL file then it switches to a different mutation. In this work we call this mutation: fuzzy mutation.

Two implementations will be discussed below: the so called constant probability implementation and an implementation based on a Gaussian distribution of probability.

4.3.2.2.2 Constant Probability

To preserve diversity, the information provided by the fuzzy rules is not always taken into consideration. To obtain this, a probability of applying the fuzzy information is used, called: *fuzzy probability*.

In the simple implementation, this fuzzy probability is constant during the run of the algorithm and it is set to be equal with the probability of mutation.

4.3.2.2.3 Gaussian Probability

The fuzzy mutation operator computes membership of the output variable. Then it computes the center of gravity approximation of this membership function. This value is referred as COG in the following. For this COG the membership μ value is computed. The current output variable is set to the value of COG with a certain probability (as we said before, we call it the *fuzzy probability*).

The *fuzzy probability* is a value that follows the right hand side of a Gaussian function. The objective is to have a high mutation probability at the beginning of the search algorithm. As the algorithm progresses (x from the equation below increases for each individual sent to mutation) the influence of the rules will not be so big. We have selected some values for the Gaussian function so that after 500 individuals sent to mutation the *fuzzy probability* should be equal with the mutation probability

$$f(x)_{final} = (1 - mutation\_probability) \cdot e^{\frac{-(x)^2}{2 \cdot (150)^2}} + mutation\_probability$$

The range of the values of this function is between (*mutation_probability*, 1]. We have discovered from our experiments that for good results diversity must be preserved. A probability of 1 will not generate a diverse population. All the individuals will tend to respect the preference of the designer as he/she described it using the fuzzy rules. To avoid this, the function is multiplied by 0.8.

The membership μ value of the value obtained after defuzzification is used here. We are using it as a measure of confidence. If the membership value is low then

it means we are in between intervals and the rules were contradictory (e.g. one wants to make the cache big, the other one wants to make it small). In this situation we decided to lower the probability to use the fuzzy information. The final probability is obtained using the following formula:

$$\text{fuzzy prob} = 0.8 \cdot \mu \cdot \left[(1 - mutation\_probability) \cdot e^{\frac{-(x)^2}{2 \cdot (150)^2}} + mutation\_probability \right]$$

4.3.2.2.4 Random Defuzzifier

There are special situations when the center of gravity based defuzzification methods do not provide good results. Such a case might happen when the input fuzzy functions have an almost rectangular shape. In this situation any value of the input will have a membership value equal (or almost equal) with 1. If the membership of the input is always 1, then the membership function from the consequent will be identical for all the inputs. This means that COG will return the same value each time. To avoid such situations we have developed a new defuzzifier inside jFuzzyLogic library. We called it: *RandomDefuzzifer*. This new defuzzifier chooses randomly a value from the output membership function but only if the intervals where the membership value is above a configurable threshold.

This defuzzifier has been used by us in Paragraph 5.5 where the membership functions are generated automatically and their shape is not trapezoidal.

4.3.2.3 Virtual Parameters

Rules provided by the computer designers are usually quite general (see 6.1 for more details about the SLVP): IF L1_Data_Cache IS *big*/*small* THEN SLVP_size IS *small/big*.

The problem arises when the level 1 data cache size is not determined by a single parameter. The size can be determined by a couple of parameters: block size, number of lines, associativity. This means that the rule might have to be split into several rules. But defining the membership functions becomes very difficult. If a parameter is “big” but all the others are small in the end the cache size is small. Defining rules becomes harder.

To solve situations like this one, we have implemented "virtual parameters". These virtual parameters are formed using a combination of other real parameters.

"In theory, there is no difference between theory and practice.
But, in practice, there is."

Jan L.A. van de Snepscheu

5 Multi-objective Hardware-software Optimization of the Grid ALU Processor

In this chapter we present the results obtained with a superscalar processor developed at the Augsbug University and the associated code optimization tool. These represented the first real tests for FADSE and many of the features developed were requirements posed by these tools.

This work is the result of collaboration between us and the Augsburg University through Professor Theo Ungerer and PhD student Ralf Jahr. The obtained results were published in conferences [28] [29] [30] (submitted) or presented at workshops [31].

5.1 GAP and GAPtimize Overview

5.1.1 Description

The Grid ALU Processor (GAP) is a single-core processor architecture developed to speed up the execution of single threaded programs. One big advantage of GAP is that it uses Portable Instruction Set Architecture (PISA) derived from a MIPS instruction set architecture. This means that GAP is able to run existing programs without any modification required. More details about the GAP processor can be found in [32] and [33].

Table 5.1-1 Parameter space for GAP

	Description	Domain
C_r	Array: rows	{4, 5, 6, 7, ..., 32}
C_c	Array: columns	{4, 5, 6, 7, ..., 31}
C_l	Array: layers	{1, 2, 4, 8, ..., 64}
C_{c1}	Cache: line size	{4, 8, 16}
C_{c2}	Cache: sets	{32, 64, 128, ..., 8192}
C_{c3}	Cache: lines per set	{1, 2, 4, 8, ..., 128}

The varied parameters and their domains are presented in Table 5.1-1.The size of the space is over 1 million (1016640) possible configurations.

In conjunction with GAP, we used GAPtimize a post-link code optimization tool developed especially for this processor. GAPtimize works on statically linked executable files compiled with GCC for PISA. Some of the code optimizations implemented are: predicated execution, a special scheduling technique, inlining of functions [28], a software-assisted replacement strategy for the configuration layers supported by code annotations called qdLRU [34], static speculation [35].

5.1.2 Objectives

We have focused on two objectives for the GAP architecture: speed - described in terms of cycles per instruction (CPI) or cycles per reference instruction (CPRI) – and hardware complexity. The second objective we used is called complexity. This metric tries to give a comparable number that describes the hardware complexity of the GAP architecture. Together with Ralf Jahr we proposed this metric because GAP does not have a hardware implementation yet. Its purpose is to compare the hardware complexity GAP configurations.

More details about these metrics can be found in our article presented at HPCS 2011 International Conference [28].

5.2 Automatic DSE on the Hardware Parameters

This work continues the design space exploration performed by Basher Shehan from University of Augsburg for his PhD thesis [36]. He has performed an extended DSE exploration and we try to see if better results can be obtained automatically.

5.2.1 Methodology

We are using NSGA-II algorithm for the DSE process. We used a crossover probability of 0.9, mutation probability was set to 1/(number of parameters) = 1/6 ~0.16. These values were recommended in [10]. Bit flip mutation, single point crossover and the binary tournament selection, proposed in [10], were used as operators. The population size was varied, to study its influence on the obtained results, from 12 to 100 individuals.

We ran up to 200 generations but the results do not improve greatly after 50 generations. The results presented in this chapter are obtained after 50 generations.

We selected 10 of the 14 benchmarks used by Shehan to reduce the time needed to evaluate an individual of the design space. The 10 benchmarks are: *dijkstra, qsort, tele-adpcm-file-decode, stringsearch, jpeg-encode, jpeg-decode, gsm-encode, gsm-decode, rijndael-encode, rijndael-decode.*

The results were obtained using 9 Intel Dual Core computers organized in a LAN located at the University "Lucian Blaga" from Sibiu, Romania. Other results were obtained using a virtual machine with 32 cores hosted by a supercomputer located at the University of Augsburg, Germany. These machines were used to obtain all the results for GAP and GAPtimize presented during this chapter.

5.2.2 Results

In our first experiments we studied the influence of the population size on the results. We changed the population size to 12, 24, 50 and 100. Good results were obtained for a population of size 50 and 100, so only they will be presented.

We ran with each population size two times and computed the average hypervolume. We ran for 50 generations with a population of 100 and 100 generations with a population of 50 individuals. Comparing them using the generation count is not fair since at generation 10, for example, the run with a population size of 50 produces far less individuals than the run with a population size of 100. We computed the number of unique individuals produced by each run (reuse is taken into consideration). We plotted the hypervolume against the average number of simulations (see Figure 5.2-1). From this figure we can conclude that even if the run with a larger population sends more individuals to simulation it finds better results in the same amount of time. We further analyzed the results and concluded that a run with 100 individuals requires 30 generations to simulate the same amount of individuals as a simulation with 50 individuals over 100 generations. This means that the run with a population with a size of 100 individuals has a larger ratio of unique individuals produced at each generation.

We analyzed the Pareto front approximation and saw that the run with a population of 50 did not (sufficiently) explore the area where the complexity is very low.

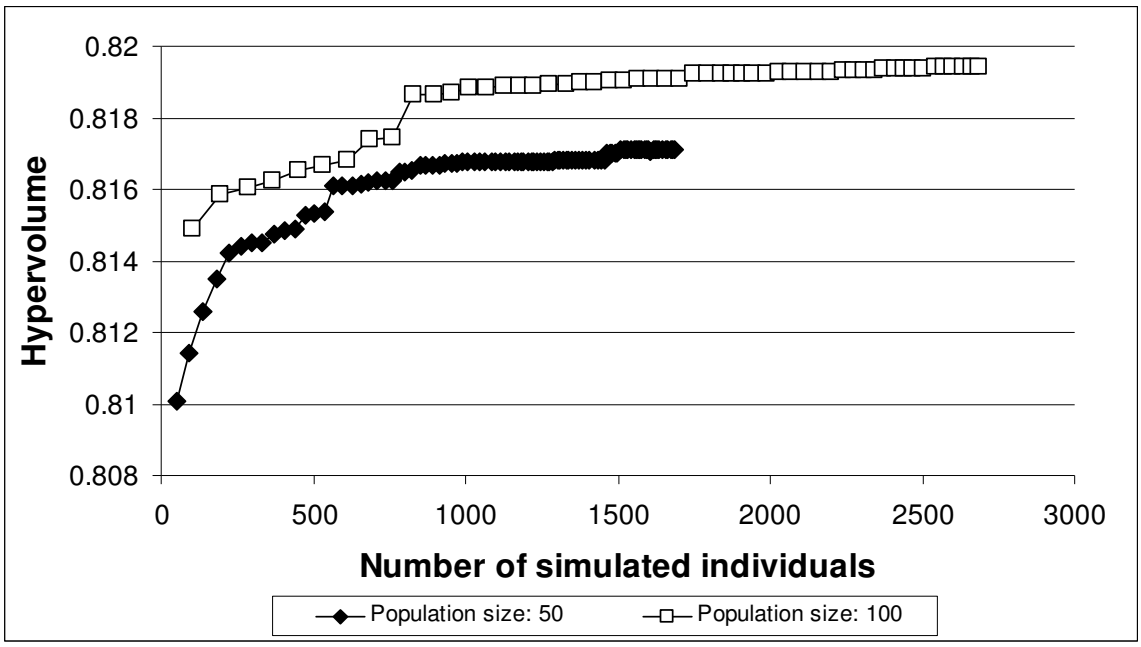

Figure 5.2-1 Average hypervolume comparison between runs with a population size of 100 and runs with a population size of 50

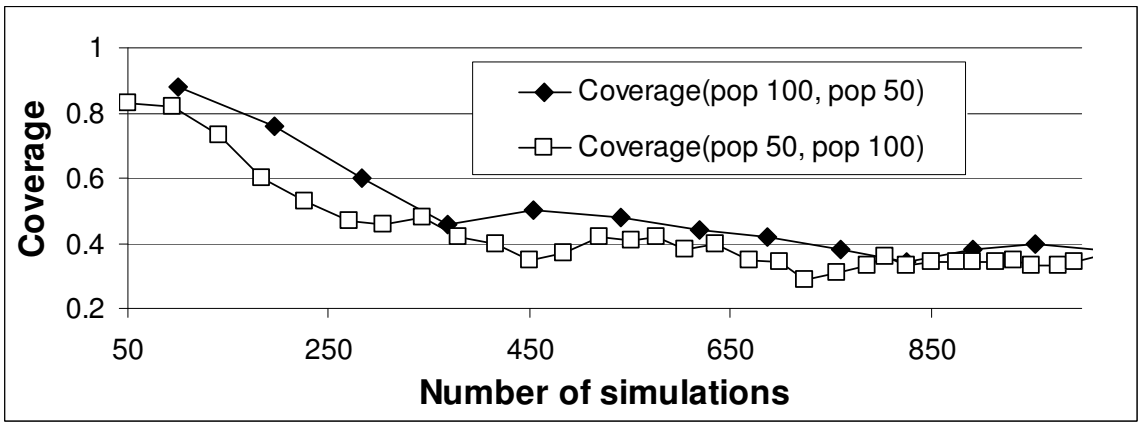

Figure 5.2-2 Coverage comparison between a DSE process with a population of 100 and another one with a population of 50

We decided to compare the two runs using the coverage metric (see Paragraph 2.4.3). According to the coverage (see Figure 5.2-2), the run with a population size of 100 individuals has slightly better results.

For the rest of the experiments we used the population size set to 100 individuals. In Figure 5.2-3 we compare the results obtained by FADSE against the manually obtained results in [32]. In Figure 5.2-4 only a section of the total Pareto front approximation is depicted. Better results were obtained by FADSE than the ones obtained during the manual exploration. FADSE was able to find configurations having half the value of complexity at the same CPI.

After analyzing the results, we observed that the rule of thumb (number of columns equal with number of rows) used in the manual exploration was not beneficial. The array of FUs had too much columns. We can conclude that FADSE can help the designer find better configurations.

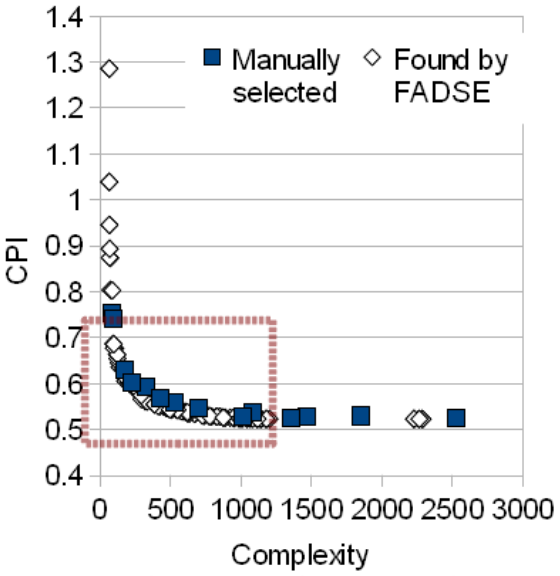

Figure 5.2-3 Total result space

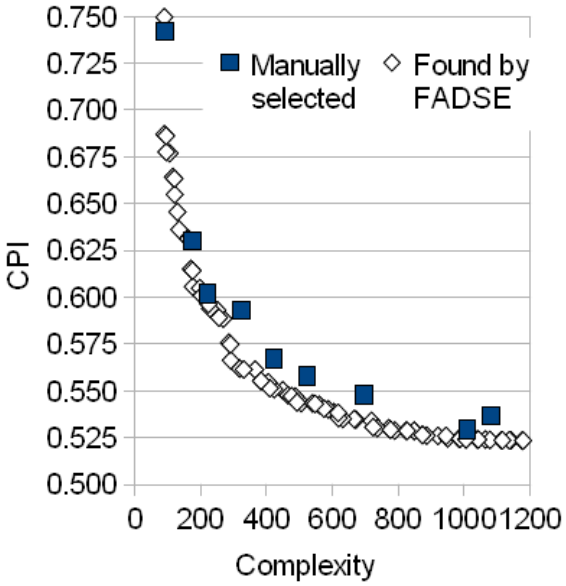

Figure 5.2-4 Zoom on most interesting area (highlighted box in Figure 5.3-3)

In Figure 5.2-3 the shape of the obtained Pareto approximated front tells us that configurations with a complexity above 1000 do not lead to a much higher performance. At this complexity, the configuration of GAP is: a matrix with 32 lines, 11 columns and 32 configuration layers, a cache of 512kB.

From the reuse point of view we can see in Figure 5.2-5 that the algorithm produces during the first generations many new individuals (i.e. individuals that were never generated before) and many of them are added to the population. As the algorithm advances, less and less new individuals are created (the rest are individuals that were already produced in the past) and, of course, even fewer of them survive to the next generation. Because of this, the reuse from the database is high. In a run with 100 generations we have reached a reuse of 67%. This means a great time reduction for the DSE process.

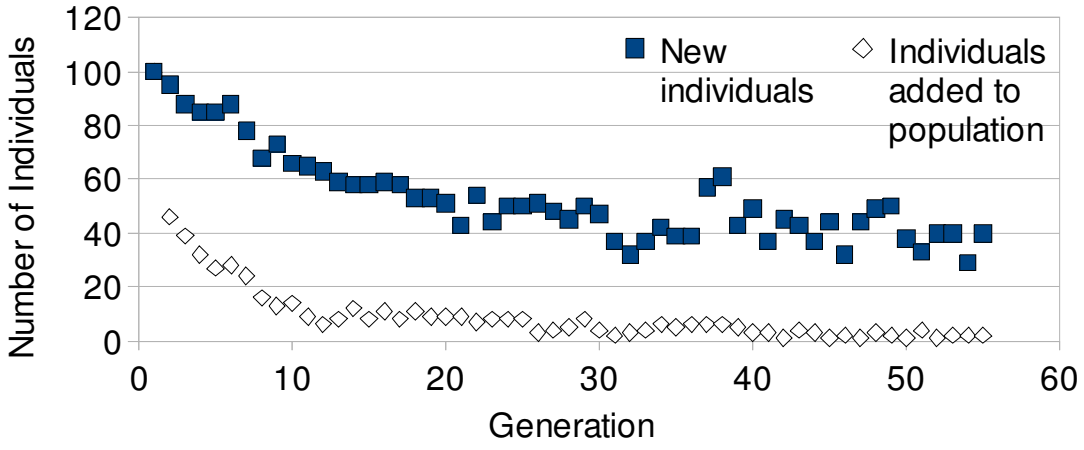

Figure 5.2-5 Comparison between the number of newly generated individuals (offspring) and the number of them that actually reach the next generation

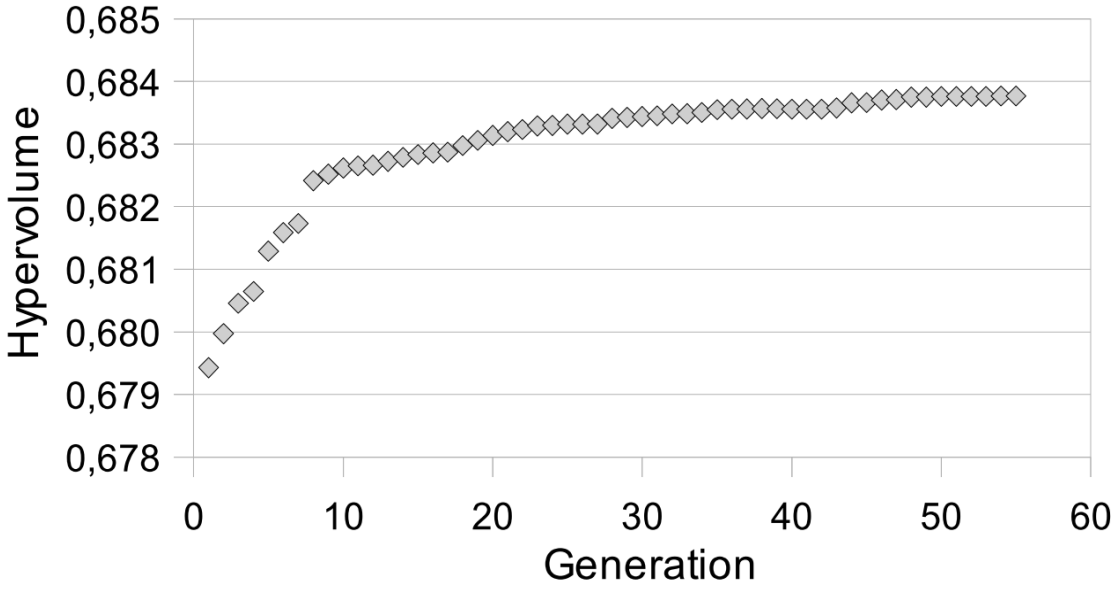

Figure 5.2-6 Evolution of the hypervolume value over the generations

Figure 5.2-5 is correlated with Figure 5.2-6 where the evolution of the hypervolume value is presented. Since fewer individuals are accepted in the next population, the hypervolume value stops increasing. We can observe that the algorithm progresses fast during the first generations but then it converges to an approximated Pareto optimal front.

5.2.3 Conclusions

With this experiment we have demonstrated that:

- FADSE can find better solutions than the human designer;
- FADSE can cope with the large design space;
- It is reliable since it was able to run for long periods of time on different systems: LANs and virtual machines with many cores;
- The database integration accelerates the DSE process considerably (67% reuse).

After careful analysis of the obtained results we concluded that GAP is a scalable architecture and that bigger caches do not cancel the effects of the ALU array.

5.3 Automatic DSE on the Hardware and Compiler Parameters

The increasing complexity of processor architectures makes it harder even for code optimization tools to find good parameters. Settings of the code optimizations might have to be substantially different for similar target platforms. We decided to use FADSE to solve this problem too.

From the code optimizations included in GAPtimize we focused only on function inlining. The main challenge is choosing the right function callers which shall be replaced by copies of the function body.

5.3.1 Methodology

We are using the same settings for the parameters as in previous section:

- bit flip mutation with a probability of 0.16 (we fix the cache parameters and the number of configuration layers);
- single point crossover with the probability to apply set to 0.9;
- binary tournament selection;
- population size 100.

GAPtimize alone has a size of the space of $2.1*10^{10}$. Together with GAP the size of the design space is over $2.1*10^{16}$. We observed, from previous experiments, that function inlining accelerates GAP because it makes the instructions accessible faster. This is due to the fact that instructions already reside in the cache or in a configuration layer. A configuration of GAP with a large cache or with many configuration layers might not benefit so much from the function inlining optimization. Thus, we decided to restrict the cache size to 8kB and the number of configuration layers to 8. The size of the space with these restrictions is ca. $1.8 *10^{13}$.

5.3.2 Results

Our first test was to see if FADSE can cope with code optimizations. We decided to fix the hardware to a matrix with 12x12 functional units and optimize at first one single objective (CPRI). We selected *dijkstra* benchmark for this experiment.

FADSE was able to find good parameters for function inlining and the execution time of the best found individual was reduced by 9.1% compared with the run with no optimization.

Next, the number of benchmarks was increased to 10, but the hardware remained fixed, so we still have single objective optimization. The best set of parameters found by FADSE lead to a reduction of the execution time of 3.9%. The increase is not as significant as for *dijkstra* because not all the benchmarks are sensitive to function inlining.

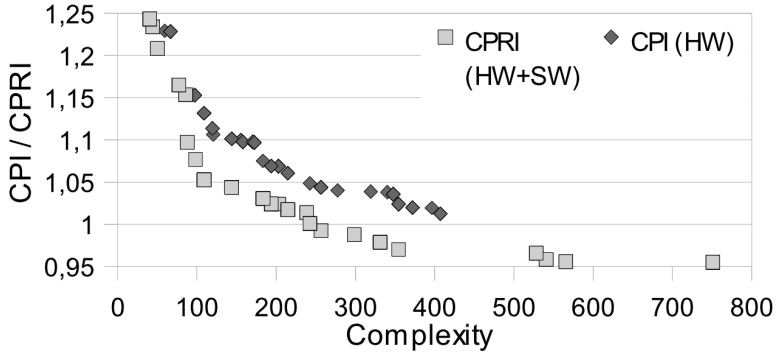

Figure 5.3-1 DSE of inlining and hardware parameters for 10 benchmarks, executed on GAP with NxNx8 array and 8 kb instruction cache

We moved then to the true DSE process, were both hardware and compiler parameters are optimized at the same time. FADSE obtained very good results. In Figure 5.3-1 a comparison in the objective space between the results obtained with and without GAPtimize is depicted. The figure proves that a run with GAPtimize obtains better results and that FADSE is able to find good parameters for inlining.

5.3.3 Conclusions

During the single objective optimization FADSE was in connection with the heuristics and able to perform inlining as an adaptive code optimization, thus providing good results.

FADSE is also able to cope with hardware and software parameters, at the same time, and find good solutions even in a huge design space ($1.8*10^{13}$ individuals) and can be used for code optimizations.

5.4 Comparison between DSE Algorithms

The results presented in this paragraph were published in our paper called "Optimizing a Superscalar System using Multi-objective Design Space Exploration" [29].

In this paragraph we are using GAP and GAPtimize to compare different heuristic algorithms. We focus on three algorithms: two genetic (NSGA-II and SPEA2) and one particle optimization (SMPSO).

5.4.1 Methodology

We use our classical configuration of the NSGA-II algorithm: bit flip mutation with a probability of 0.16, single point crossover with the probability to apply set to 0.9, binary tournament selection and a population size of 100 individuals.

For SPEA2 we use the same operators and probabilities as for NSGA-II and we set the archive size to 100.

For the particle swarm optimization algorithms we used a swarm size of 100. For the velocity computation we used the values recommended in [13]:

- C_1 and C_2 are randomly chosen in the interval [1.5, 2.5];
- r_1 and r_2 are randomly chosen in the interval [0, 1];
- inertial weight W is fixed to 0.1.

We have generated a random population and all the algorithms were started from this population. This way, a fair comparison between the algorithms can be performed. Due to time constraints (about 5 days per run) we could not afford to run multiple times and present average results.

For the runs with GAP we decided to vary all six parameters. For GAPtimize we fixed the cache size and the number of configuration layers (see 5.3.1 for motivation). All the runs were performed on 10 benchmarks from MiBench suite.

When running with GAPtimize, the benchmarks were compiled in GCC with function inlining deactivated, because GAPtimize does this code optimization.

5.4.2 Results

5.4.2.1 Results on GAP

We began by comparing the three algorithms using only GAP. We ran all the algorithms up to generation 50. We computed the coverage for all the possible combinations. The first comparison was made between NSGA-II and SPEA2 (see Figure 5.4-1). Their results are similar for the first generations but then NSGA-II finds more individuals that dominate individuals obtained by the SPEA2 algorithm. The conclusion, we can draw from the coverage value over the generations, is that NSGA-II performs better than SPEA2.

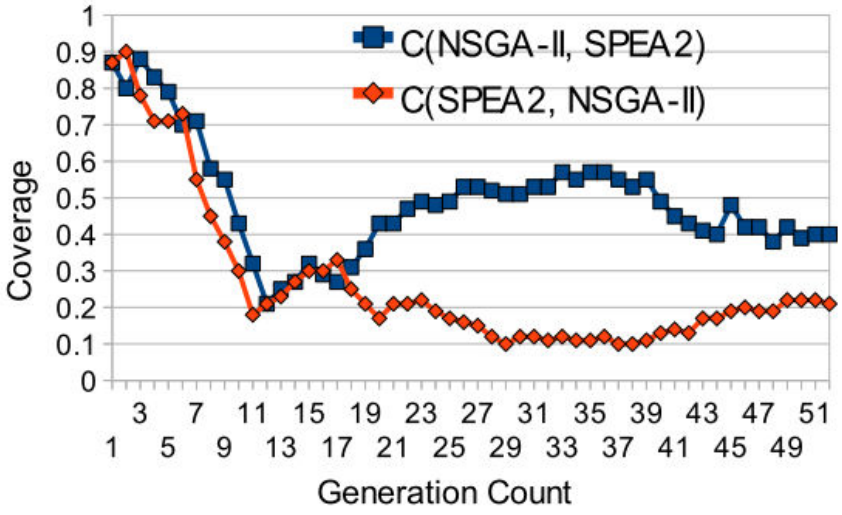

Figure 5.4-1 Coverage comparison between NSGA-II and SPEA2

Next we compared NSGA-II and SMPSO. In Figure 5.4-2 is illustrated the coverage value over the generations. SMSPO generates more individuals that dominate individuals found by NSGA-II.

SMPSO seems to be the best algorithm from the coverage point of view. On both comparisons SMPSO has a great advantage over NSGA-II and SPEA2 during the first generations. This means that SMPSO converges faster to better solutions, as for NSGA-II and SPEA2 it takes more time to discover individuals of similar quality.

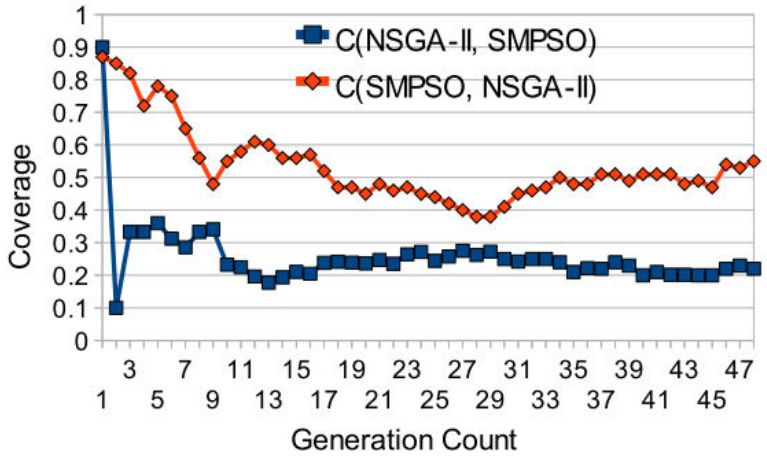

Figure 5.4-2 Coverage comparison between NSGA-II and SMSPO

We compared the algorithms from the perspective of a different metric: hypervolumne. The evolution of the hypervolume over the generations is shown in Figure 5.4-3. The hypervolume gives information about the convergence of the algorithm. When the same reference point is used (see Paragraph 2.4.1), it can also give us a hint about the quality of results. We are using the same reference point so we can say from Figure 5.4-3 that SMPSO finds the best results. SMPSO reaches a hypervolume that is never reached by the two genetic algorithms.

From a convergence point of view, the particle swarm optimization algorithm is again the best. It converges faster than the genetic algorithms. SPEA2 has a better start than NSGA-II but the latter obtains a better hypervolume value in the end.

In the domain of computer architecture we are dealing with long evaluation times because simulators are used. Therefore it is important to count how many simulations each algorithm requires to reach a certain hypervolume. If an algorithm

generates the same individuals over and over again they can be reused from the database. Reusing the individuals means fewer simulations, thus less time required for the exploration time.

First, we present the number of simulations performed by each algorithm to reach a certain generation (see Figure 5.4-4). We can see that the SMPSO algorithm performs more simulation compared to the genetic algorithms. NSGA-II does more simulations than SPEA2. Thus, to reach a certain generation, SMPSO and NSGA-II needed to perform more simulations.

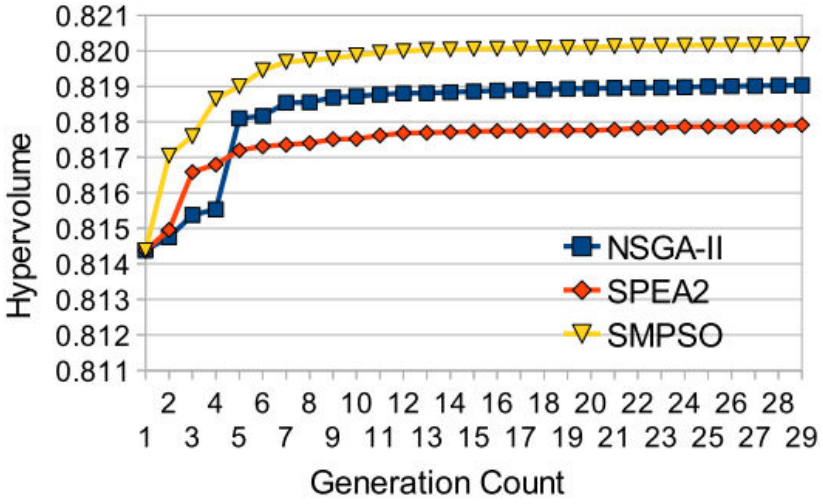

Figure 5.4-3 Hypervolume comparison between NSGA-II, SPEA2 and SMPSO

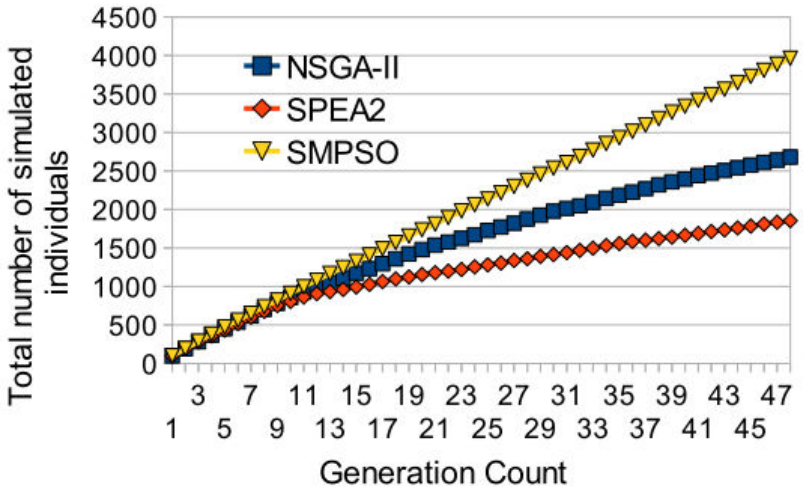

Figure 5.4-4 This figure shows how many individuals were simulated to reach a certain generation.

With this in mind we decided to compare the hypervolume from this point of view. In Figure 5.4-5, we plot how many individuals had to be simulated to reach a certain hypervolume. The ranking of the algorithms does not change. Even if the SMPSO algorithm requires more simulations, the quality of the obtained results justifies the effort.

Figure 5.4-4 gives us data about the reuse percentage. During 50 generations (5000 evaluated individuals) SMPSO sent for evaluation ca. 4000 individuals. NSGA-

II and SPEA2 sent ca. 2700, 1800 respectively. This means a 20% reuse for SMPSO, 46% for NSGA-II and 63% for SPEA2. The reuse method incorporated into FADSE leads to a huge time reduction for the DSE process.

SPEA2 tends to produce the same individuals again because it retains more duplicates in the archive than the NSGA-II algorithm. We already presented the reason for this in 2.1.4. SMPSO produces new individuals so often because all the particles are moved at each generation. SMSPO applies a sort of mutation on all the parameters (fly), not on only 1/(number of parameters) of them.

Because SPEA2 retains many duplicates, the spread of solutions in the objective space is not so good, compared with NSGA-II and SMPSO.

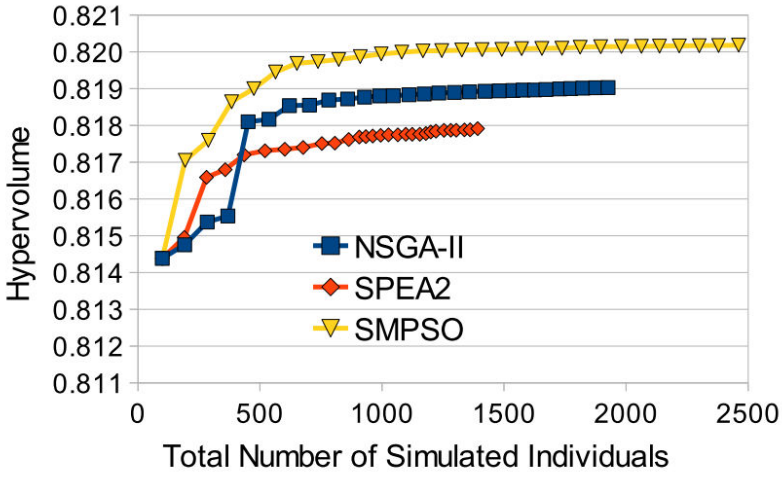

Figure 5.4-5 Hypervolume comparison of the three selected algorithms against the total number of evaluated designs

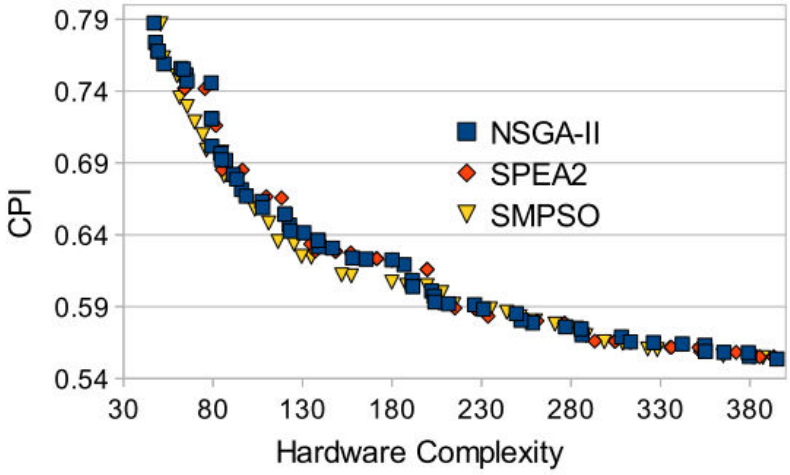

Figure 5.4-6 Section of the Pareto front approximations obtained by each algorithm after 50 generations

We compared the Pareto fronts approximation found by the algorithms (see Figure 5.4-6). The figure presents only a section of the entire Pareto front. We decided to depict only this area because on the rest of the front there are almost no differences between the algorithms.

It can be observed that SMSPO finds slightly better results on a small area of the Pareto front approximation between the complexities 100-180, while the complexities of all the solutions found by the algorithms range from 30 to 2000. Even

in this small area, the differences are minimal. The metrics might be misleading; showing a big difference between the algorithms, while in reality the difference is only marginal. Even so, the fact that SMSPO converges faster recommends him as a good algorithm for design space exploration.

We decided to use the Two Set Difference Hypervolume metric (see Paragraph 2.4.2) to see how much of the space is really dominated by a single algorithm. The results can be seen in Figure 5.4-7. The highest value is obtained when we are comparing the SMPSO algorithm with SPEA2. The next two (in terms of value) comparisons are between SMPSO and NSGA-II and between NSGA-II and SPEA2, so this metric keeps the ranking showed by the other metrics. It also shows us that in fact only 0.001-0.002% of the entire hypervolume covered by the approximated Pareto fronts is dominated by the winning algorithms.

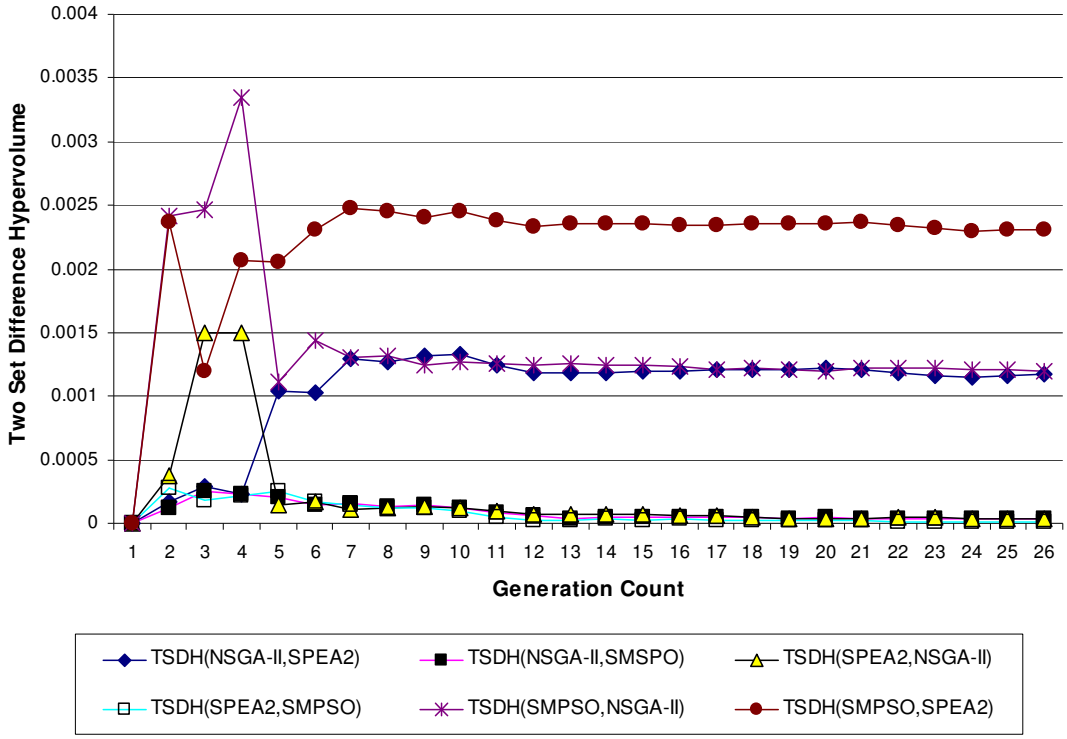

Figure 5.4-7 Comparison between the algorithms using the Two Set Difference Hypervolume (TSDH) metric

5.4.2.2 Results on GAP with GAPtimize

In this chapter we compared the three algorithms on GAP with GAPtimize. The same 10 benchmarks were used as in previous section.

We start with a coverage comparison between the two genetic algorithms. For the first 6 generations there is no much difference between the algorithms (see Figure 5.4-8). From there on SPEA2 gains a huge advantage. By the end of the DSE process NSGA-II does not dominate any of the individuals found by SPEA2, while SPEA2 dominates over 60% of the individuals discovered by NSGA-II. We selected SPEA2 as the best algorithm and compared it with SMPSO using the coverage metric (see Figure 5.4-9). SMPSO has the best results, but the difference is not that big, as we have seen between SPEA2 and NSGA-II.

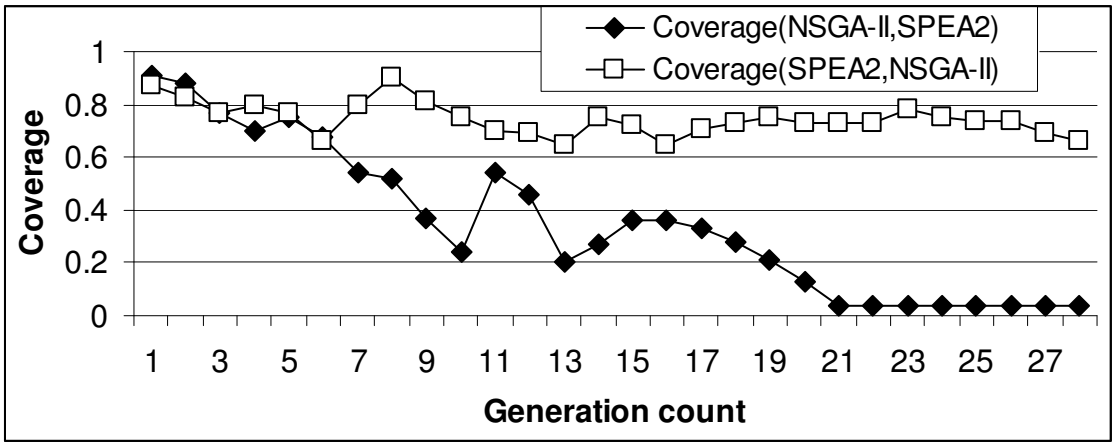

Figure 5.4-8 Coverage comparison between DSE runs with NSGA-II and SPEA2

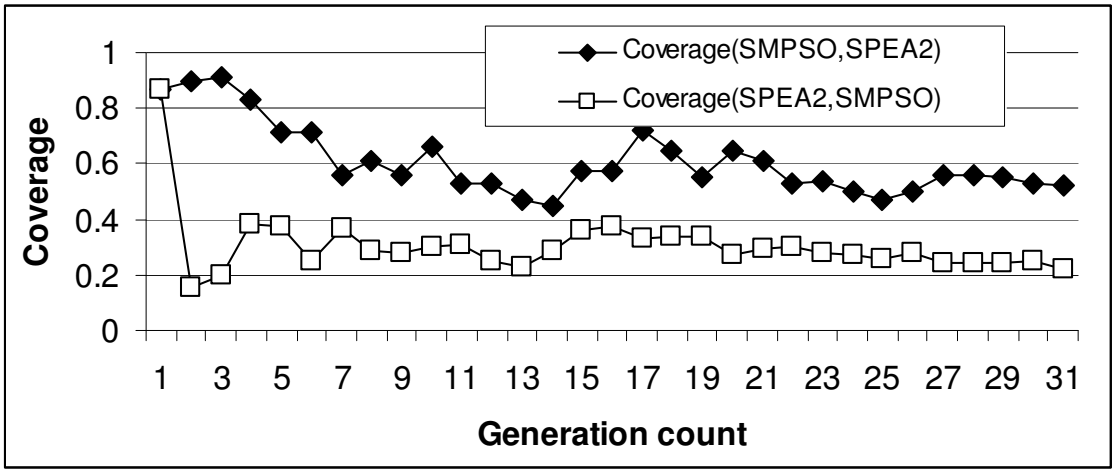

Figure 5.4-9 Coverage comparison between DSE runs with SPEA2 and SMPSO

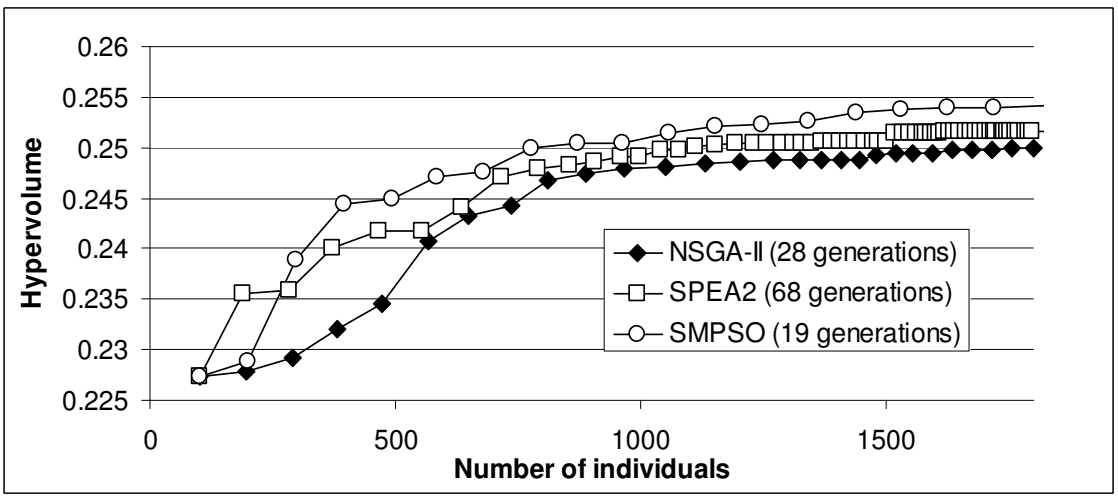

Figure 5.4-10 Hypervolume comparison between NSGA-II, SPEA2 and SMPSO considering the number of simulated individuals

In this experiment, SPEA2 simulates least individuals to reach the same generation. NSGA-II performs around 1800 simulations in 29 generations. SMPSO generates the same amount of individuals in only 19 generations, while SPEA2 requires 68 generations to reach that number. It must be noted that, after the 20th generation, the number of new individuals produced by SPEA2 decreases

dramatically (20 out of 100 individuals at generation 20 to 5 out of 100 at generation 69).

With this in mind we compared the hypervolume values (see Figure 5.4-10). SMPSO is still the best and also has the fastest convergence speed from the three.

We analyzed the evolution of the Pareto fronts approximation (not showed here) and we concluded that during the generations the difference is not very big between the algorithms in terms of quality of solutions, as it was depicted by the coverage metric.

Then, we compared the Pareto fronts approximation obtained by the three algorithms after around 1800 simulations (see Figure 5.4-11). There are not many individuals from other algorithms except SMPSO, in the figure, because they are overlapped. The algorithms obtain practically the same results.

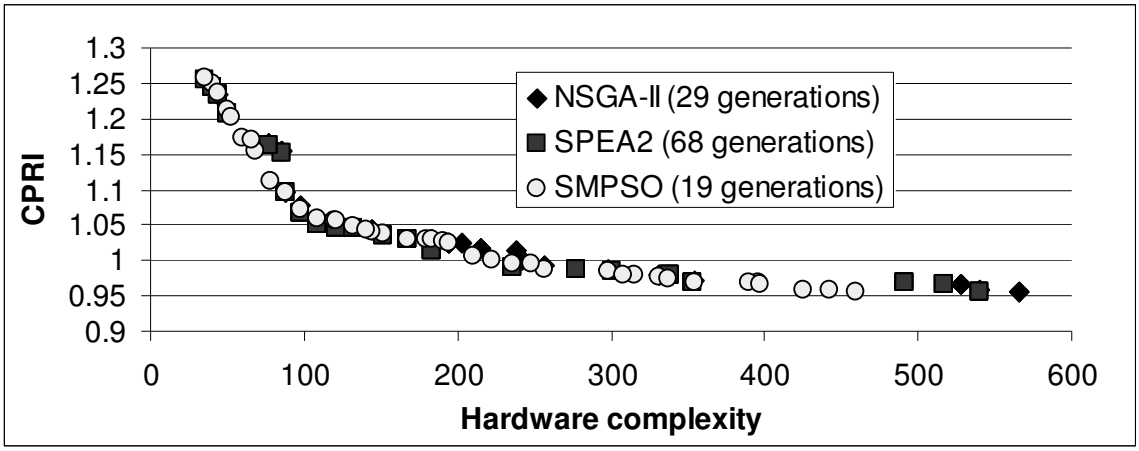

Figure 5.4-11 Final Pareto front approximations obtained by NSGA-II, SPEA2 and SMPSO after 1800 simulations

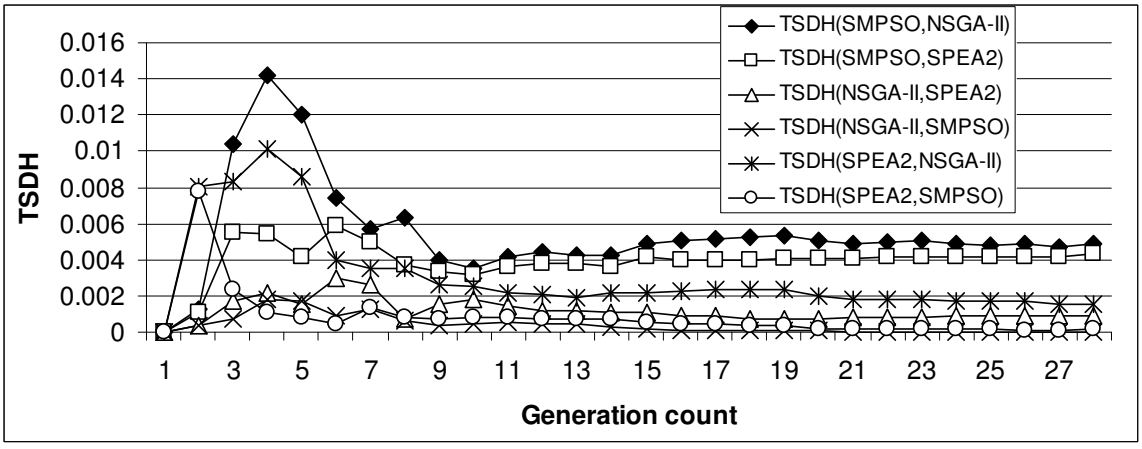

Figure 5.4-12 Two set difference hypervolume (TSDH) comparison between the algorithms

Finally we compare the three algorithms (two by two) using the two set difference hypervolume metric – TSDH – (see Figure 5.4-12). The results only confirmed our conclusions. The difference between the results obtained by the algorithms is very small. For the first generations (until generation 7) SMPSO has better results revealed by a higher TSDH value, especially against NSGA-II. SPEA2 also obtains better results compared to NSGA-II for the first generations.

5.4.3 Conclusions

Analyzing the Pareto front approximations obtained by the three algorithms we can conclude that the differences are small. The biggest differences are not greater than 1% in terms of CPI, and usually reside between 0.1% and 0.01%, at the same complexity. The difference between the algorithms comes in convergence speed.

From the convergence speed point of view, the particle swarm optimization algorithm preformed the best, with a very high convergence speed. This confirms the results obtained in [13]. From the two genetic algorithms NSGA-II converged faster and found better solutions on GAP, but when GAPtimize was added SPEA2 performed better, both in terms of convergence speed and quality of solutions. In terms off spread of the Pareto fronts approximations, SPEA2 was clearly the worse, with many duplicates in the final population (archive).

An important conclusion we can draw from these experiments is that the coverage metric can be misleading. It can show a big difference between the algorithms while in real terms that is not the case. The problem is that it has no threshold for domination, even if the difference on one objective is very small the individual is still considered dominated. Coverage with such a threshold must be considered for real problem optimizations. Such metrics are proposed in [15]. Hypervolume does help the user to observe the convergence speed of the algorithms. It also gives some information about the quality of the solutions when multiple algorithms are shown in the same figure and a single hypervolume reference point is used. However, the difference between the values of the hypervolume should not be considered. Hypervolume two set difference tries to give a little more information about how much of the space is actually dominated by a single algorithm, but it can also be misleading because the values are dependent on the position of the hypervolume reference point and can not be taken into consideration very seriously. Because the true Pareto front is unknown, finding a metric that correctly compares algorithms is difficult. We recommend that the user should look at the obtained Pareto fronts and draw a conclusion based on his/hers experience.

5.5 Automatically Generated Rules from Previous Exploration

This work extends the fuzzy logic integration with FADSE. The purpose is to obtain the rules automatically from previous simulations. The idea is that a user can run a design space exploration process on a single short benchmark. Obtain results for that benchmark, extract rules and apply these rules on the design space exploration process with multiple/long benchmarks. The second situation where this can be applied is when a company builds a processor and performs the DSE. A client might come and want to optimize that architecture for a specific task. Or the producer wants to add a new feature to the design. The company could extract knowledge from its previous exploration and offer it to the client so it can accelerate his DSE. From this we can see two possible situations: when the data is extracted from a single benchmark and then applied to multiple benchmarks (called by us “special to general”) and when data is extracted from multiple benchmarks and applied to a specific task.

The results from this paragraph are from our article "Boosting Design Space Explorations with Existing or Automatically Learned Knowledge" [30].

5.5.1 Methodology

Ralf Jahr's idea was to use machine learning techniques to calculate decision trees. These trees were then translated into rules. This way a feed-back loop to automatically make use of results obtained in prior DSE runs was created.

We performed tests with both implementations for the fuzzy mutation (see 4.3.2.2) and concluded that better results are obtained with the Gaussian distribution of probability to use the information provided by the rules.

With this system, rules for the M-SIM simulator (see Chapter 6 for more details) were generated. We had good results in terms of finding the rules, but they were not used during a DSE process.

For FADSE we use a similar configuration as in previous chapters: population size of 50 individuals, single point crossover with a probability of 0.9, fuzzy bit flip mutation with a Gaussian probability to apply the fuzzy rules, the mutation probability is set to 0.16. We are using the same selection of 10 benchmarks from the MiBench suite.

5.5.2 Results

In our first test we start from a single benchmark, extract the rules and then run with these rules on all the benchmarks. As the single benchmark we have selected *stringsearch*. This benchmark is one of the smallest in the MiBench suite.

With the selected benchmark, we performed a DSE and obtained 1100 unique individuals. We used the classification method presented in Chapter 5.5.1 and extracted the rules.

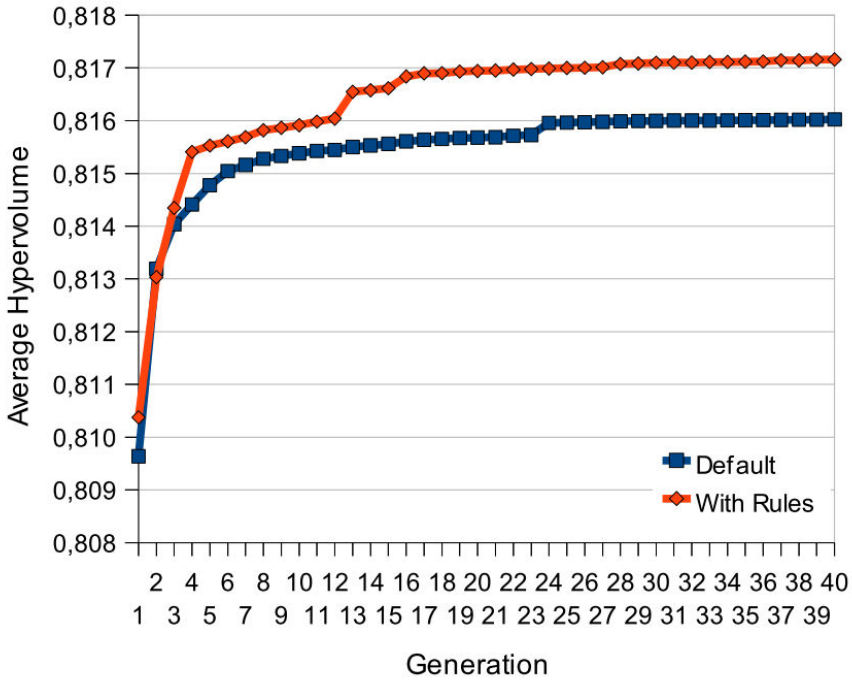

Figure 5.5-1 Special to general experiment

With the obtained rules we run the full design space exploration (10 benchmarks). To provide a fair comparison we ran with and without the rules five times up to generation 40. Each time we start from a random initial population.

In Figure 5.5-1 the average hypervolume obtained by the runs with and without rules is presented. The run with rules obtains better results: it converges faster

and also reaches a hypervolume value higher than the one obtained by the run without rules.

With this technique we have obtained a 50% reduction in the time required to obtain the same quality of results. The DSE process with a single benchmark is very quick. Then the run with the extracted rules converges much faster: at generation 11 the hypervolume obtained by the run with rules is equal with the hypervolume obtained without rules after 24 generations.

The next possible situation is to start from a previous exploration and then optimize the architecture for a certain benchmark. We used these rules to optimize GAP for two application domains: one for image encoding/decoding with JPEG, the other is encrypting/decrypting data with the *Rijndael* algorithm (AES) from the MiBench suite.

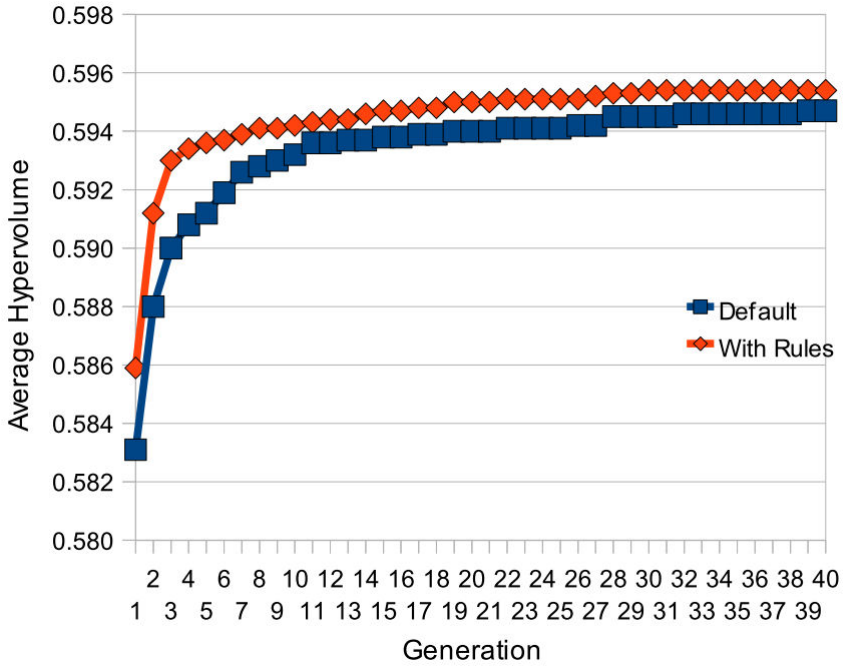

Figure 5.5-2 Hypervolume JPEG

For both benchmarks FADSE is ran for 40 generations five times.

The average results for JPEG are shown in Figure 5.5-2, for *Rijndael* in Figure 5.5-3. The obtained results are again better than the ones obtained without rules. The results are especially good for JPEG. The hypervolume enclosed by the individuals found during the DSE process is higher with rules than without for all the generations. Using rules leads to a faster convergence, and also the quality of the results obtained with rules is never achieved (during the 40 generations) by the runs without rules. From the convergence point of view, we can see in Figure 5.5-2 that the hypervolume achieved with rules after 3 generations is achieved without rules only after 11 generations. In our situation a generation can last for about 3-4 hours, but with other simulators this might mean days of simulation saved.

For *Rijndael*, the start is from a lower hypervolume, but with rules it manages to overcome this drawback and to surpass the runs without rules. Again the hypervolume reached for *Rijndael* with rules is never reached by the run without rules.

Experiments with a constant probability to apply fuzzy rules were also conducted, but the results were not so good. It seems that the high number of

available rules, combined with many membership functions associated to each parameter, leads to very good results and helps maintaining the diversity. In Chapter 6 we observed that runs with constant probability work better when the rules do not cover so much of the available parameters.

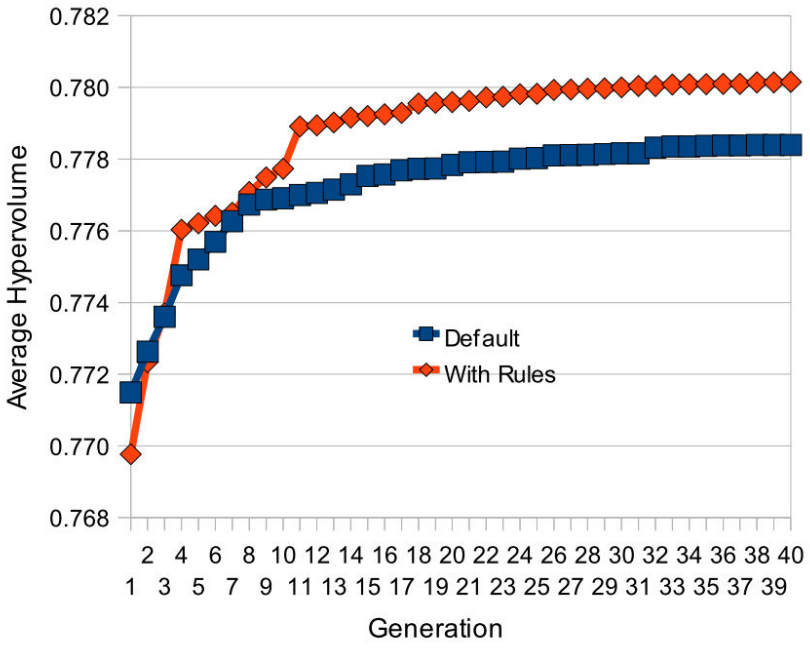

Figure 5.5-3 Hypervolume Rijndael

5.5.3 Conclusions

In this chapter, we proved that the fuzzy rules interface provided by FADSE can be extended to use other interesting ideas. Rules are automatically generated from previous explorations using data mining techniques and injected into FADSE through the fuzzy rule system. This approach might be useful in several situations: when we can run a single benchmark, extract the rules and then apply the learnt information in a broader (more time consuming) design space exploration. The other situation is when the architecture has already been optimized for general applications but we want to optimize it for a specific application. The information gained from the previous exploration can be used and it will accelerate considerably the DSE process.

5.6 Running with Hierarchical Parameters

When combining multiple code optimization passes, one has to keep in mind that the design space to explore grows dramatically making it very hard to find very good solutions. The design space increases even more if also the order of the passes is considered. Nevertheless, it is possible that combinations of optimizations lead to even higher performance gains compared to the performance gains of the individual optimizations. Hence it is important to incorporate optimization techniques as already mentioned at the start of this chapter. Beyond this, when analyzing the parameter vector consisting of all parameters, one will come to the conclusion that it is a common approach to have flags turning optimizations on and off. These flags have an important role, as they can remove any influence and importance of the parameters for an optimization if it is disabled. Hence the algorithm for the DSE should keep this in

mind and not generate on and on individuals with different "genes" but the absolutely same "phenotype", i.e. the same program binary.

In the case study all three optimizations (function inlining, static speculation, qdLRU) can be turned off an on.

5.6.1 Methodology

This chapter presents preliminary results obtained with FADSE and hierarchical parameters. We started developing the hierarchical parameters especially for this experiment, but the work is still in progress at the time of writing this thesis.

We used the NSGA-II algorithm with the usual parameters: population size 100, mutation probability 1/number of parameters. Because we wanted to test the efficiency of the hierarchical parameters we ran with and without this information. We used bit flip mutation (1/number of parameters probability) and single point crossover (0.9 probability to apply it) for the classical run. When running with hierarchical parameters the difference is that we are using the special mutation and crossover operator presented in Paragraph 4.2.2. We ran the experiment for 40 generations.

One benchmark was run until now: *quick sort* from the MiBench suite.

5.6.2 Results

The preliminary results obtained for quick sort are shown in Figure 5.6-1. It can be seen that the run with the hierarchical parameters obtained better results than the run without this information.

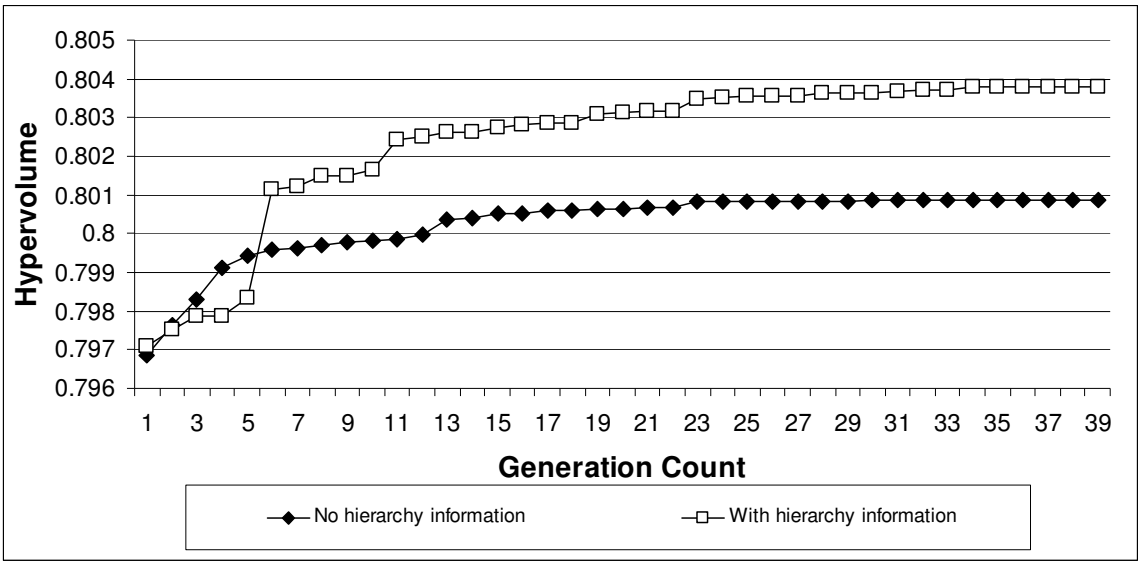

Figure 5.6-1 Hypervolume comparison between the run that considered the information about hierarchical parameters and the one that did not

More simulations will be performed, on several benchmarks to draw a final conclusion about the influence of these hierarchical parameters, but the results are promising.

"The best way to escape from a problem is to solve it."

Alan Saporta

6 Multi-objective Optimization of the Mono-cores and Multi-cores Architectures

This chapter continues the work performed by Dr. Árpád Gellért in his PhD thesis [37] and some subsequent articles [38][39]. We took his manual design space exploration performed on a Simultaneous MultiThreaded (SMT) architecture and extended it to multiple parameters with the use of FADSE. Afterwards, we moved to multi-core architectures as well.

We try to improve the results obtained by FADSE through different methods of including domain-knowledge in the DSE algorithms.

6.1 M-SIM Simulator Overview

M-SIM [40] is a cycle accurate processor simulator based on SimpleScalar 3.0d [41]. It allows multi-threaded [42] micro-architectural simulation. The target architecture is a superscalar Alpha AXP 21264.

M-SIM integrates the Wattch framework [43] for power consumption estimation. It is able to run benchmarks from suites like: SPEC 2000 [44], SPEC 2006, MiBench, Mediabench, etc.

All our simulations were performed on the SPEC 2000 benchmarks. We have selected six integer benchmarks (bzip, gcc, gzip, mcf, twolf and vpr) and 6 floating-point benchmarks (applu, equake, galgel. lucas, mesa and mgird).

In this chapter, we are presenting evaluations on two versions of the M-SIM simulator: version 2.0 and 3.0. The difference between them is that M-SIM 3 allows multi-core simulation (only independent tasks).

The base configuration of the M-SIM simulator is depicted in Table 6.1-1.

Table 6.1-1 M-SIM baseline configuration

Execution Latencies	**Execution unit**	**Number of units**	**Operation latency**
	intALU	4	1
	intMULT / intDIV	1	3 / 20
	fpALU	4	2
	fpMULT / fpDIV	1	4 / 12
Superscalarity	Fetch / Decode / Issue / Commit width = 4		
Branch predictor	bimodal predictor with 2048 entries		
Selective Load Value Predictor (SLVP)	1024 entries, direct mapped, access latency: 1 cycle, prediction latency: 3 cycles (2 cycles L1 data cache tagging + 1 cycle SLVP access)		
Caches and Memory	**Memory unit**		**Access Latency**
	64 KB, 2-way associative L1 data cache		1 cycles
	64 KB, 2-way associative L1 instruction cache		1 cycles
	4 MB, 8-way associative unified L2 cache		6 cycles
	Memory		100 cycles
Resources	**Register File:** [32 INT / 32 FP]*8		
	Reorder Buffer (ROB): 128 entries		
	Load/Store Queue (LSQ): 48 entries		

In Árpád Gellért PhD thesis, a novel selective load value prediction (SLVP) mechanism is introduced [39]. The idea is to predict long latency instructions (loads) that miss in the level 1 data cache. Árpád Gellért and his colleagues proved that the SLVP can increase performance and reduce the energy consumption [38]. He has

performed a manual design space exploration is his work and only two parameters (number of sets in the level 1 cache and the level 2 cache) were varied from the baseline architecture, thus good configurations might not be found.

We decided to vary more parameters. They are depicted in Table 6.1-2 along with the upper limits and the lower limits we have imposed.

Table 6.1-2 Parameters of the M-SIM simulators

Parameter		Lower limit	Upper limit
DL1 cache	Sets	2	32768
	Block size (bytes)	8	256
	Associativity	1	8
IL1 cache	Sets	2	32768
	Block size (bytes)	8	256
	Associativity	1	8
UL2 cache	Sets	256	2097152
	Block size (bytes)	64	256
	Associativity	2	16
SLVP (entries)		16	8192
Decode/Issue/Commit width		2	32
ROB size (entries)		32	1024
LSQ size (entries)		32	1024
IQ size (entries)		32	1024
Number of physical register sets (int/fp)		2/2	8/8
Integer ALU		2	8
Integer MUL/DIV		1	8
Floating point ALU		2	8
Floating point MUL/DIV		1	8

These 19 parameters generate a design space of over $2.5*10^{15}$ (2.5 millions of billions) and makes an exhaustive search impossible, therefore FADSE was employed.

6.2 Optimizing M-SIM 2 Architecture

The purpose of this work is to try to find better configurations than the ones obtained by Árpád Gellért et al. Since only a few parameters were varied there, we want to prove that the SLVP scheme leads to lower energy consumption at the same CPI for other, closer to optimal, configurations. This work has been submitted to IET Computers & Digital Techniques [45].

6.2.1 Methodology

Evaluating the architecture on a single benchmark takes a couple of hours. Thus, we have decided to reduce the number of simulated dynamic instructions from 1 billion to 500 million (first 300 million instructions are skipped). This means that 24 hours are required to simulate a configuration (individual) on all the benchmarks.

Because of this change, we had to redo the manual exploration performed by Árpád Gellért in his PhD thesis. We kept the same settings: 80nm CMOS technology and a 1.2 GHz frequency for simulation. Gellért used IPC in his simulations we have switched to CPI (1/IPC) to have both objectives (speed, energy consumption) minimized.

The power modeling methodology used by the simulator is presented in more detail in [43].

For FADSE, we used the following parameters: population size was set to 100 as recommended in [10]. For mutation, we used bit flip mutation with a probability of 1/(number of varied parameters). We varied 19 parameters and as consequence we set the mutation probability to 0.05. Single point crossover was selected and the probability to apply crossover was set to 0.9 (as specified in [10]). For selection the binary tournament selection operator described in [10] was used. The mutation operator was changed for the runs with fuzzy information as described in 4.3.2.2. We limited the runs to 25 generations due to time constraints.

The number of generations was selected after analyzing the first runs taking into consideration the following stop criterion: we observed the hypervolume progress. If there was no progress for at least X generations, we considered that the algorithm has converged. To measure the progress we used the following formula:

$$Progress = \sum_{i=1}^{X} (H_k - H_{k-i})$$

where H_k is the hypervolume of the current generation k, $X \leq k$. When this sum is smaller than a specified threshold θ the algorithm was stopped.

6.2.1.1 Manual Exploration

We doubled, halve, quarter and eighth the L2 cache size. For the L1 cache size we divide it by 2, 4 and 8. The initial sizes are the ones considered in the baseline architecture (see Table 6.1-1). We are using the following notations: *m*UL2_*n*DL1 means a configuration using m*4 MB 8-way associative unified L2 cache (m=2, 1, 1/2, 1/4, 1/8) and n*64 KB 2-way associative L1 data cache (n=1, 1/2, 1/4, 1/8).

This manual exploration is not the main purpose of this experiment but the optimal configurations found are used in the following experiments.

In Figure 6.2-1, the relative CPI reduction obtained with a SLVP with 1024 entries is depicted against a configuration **without the SLVP scheme**. We can see that the L2 cache can be reduced to its half size and the level 1 cache can have a size 8 times smaller and we still gain performance over the baseline architecture. In Figure 6.2-2 we can see that this reduction of the cache size leads to lower energy consumption. From the CPI point of view, reducing the level two cache size, to more that half the original size, decreases the performance.

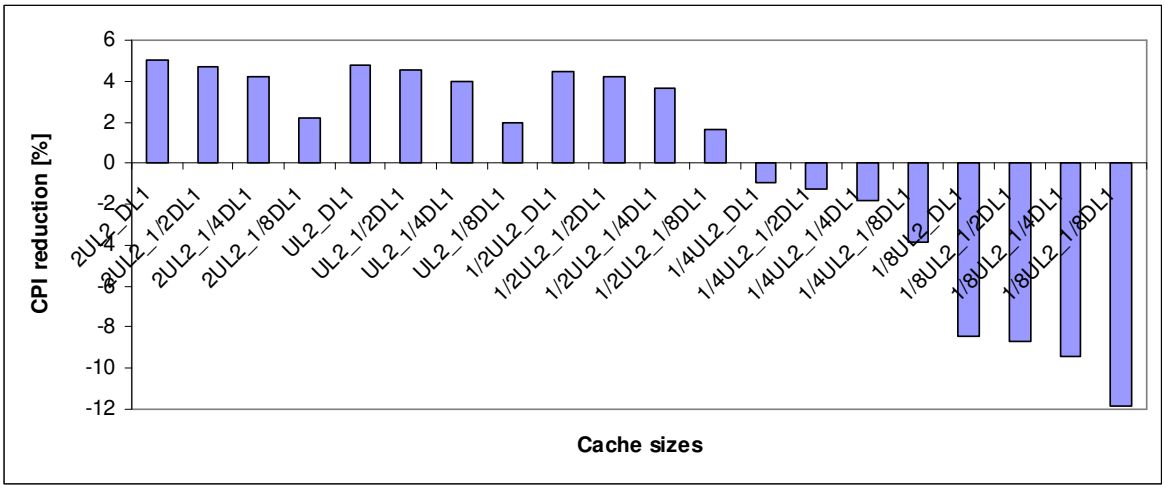

Figure 6.2-1 Relative CPI reduction reported to UL2_DL1 without SLVP as baseline

The same idea is used when we are looking at the energy reduction (see Figure 6.2-2). In this figure, the relative energy increase is shown against a configuration without SLVP. The SLVP can be used in conjunction with smaller caches and important energy reductions are obtained. It is interesting that the energy reduction is lower in the case of reducing the L2 cache to 1/8 than in the case of quartering it. This happens because when the L2 cache is reduced by a factor of 8, the static energy decreases, but at the same time, the miss rate increases which leads to a higher energy consumption. The optimal configuration from the energy point of view is: a quarter of the L2 cache (2 MB) and an eighth of the L1 data cache (8 KB).

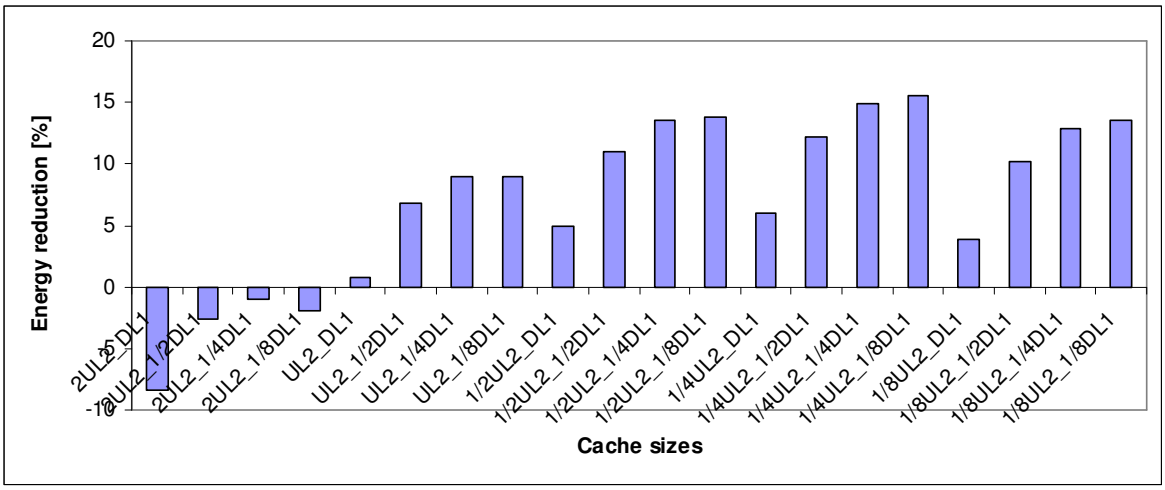

Figure 6.2-2 Relative energy reduction reported to UL2_DL1 without SLVP as baseline

From this experiment, we extracted the optimal configurations (from our point of view) we have found. The best configuration in terms of CPI is 2UL2_DL1. The best in terms of energy consumption is 1/4UL2_1/8DL1. We constructed a Pareto front with these configurations and chosen some configurations that are good taking into consideration both objectives, they are: 1/2UL2_1/2DL1 and 1/2UL2_1/4DL1. We will use these configurations in our future experiments as starting points for the automatic design space exploration.

6.2.1.2 Running Without Any Information

We first started FADSE together with M-SIM2 simulator with no prior information. FADSE was started from a random initial population. We varied the parameters described in Table 6.1-2 with the hope to find better configurations than the ones found by manual exploration. To avoid infeasible configurations (either impossible or known to provide bad results) we have used the following constraints:

$$UL2 > DL1 + IL1$$
$$UL2\_bsize \geq DL1\_bsize$$
$$UL2\_bsize \geq IL1\_bsize$$

where UL2_bsize, DL1_bsize and IL1_bsize are the block sizes for the unified L2 cache, L1 data cache and L1 instruction cache, respectively.

The size of the caches was also limited between the following borders:

DL1: 16 KB - 1 MB
IL1: 16 KB - 1 MB
UL2: 1 MB - 8 MB

The design space obtained with these restrictions had a size of $3.8*10^{13}$.

While analyzing the results (see 6.2.2) we have observed that the constraints used were too restrictive and that FADSE was not able to explore regions with low energies (smaller caches). Because of this, we relaxed the borders of the caches allowing FADSE to search in a larger space. The minimum cache capacities allowed were reduced:

DL1: 4 KB - 1 MB
IL1: 8 KB - 1 MB
UL2: 256 KB - 8 MB

The design space is reduced to 3% of the initial space, meaning 7.7 *1013 (77 thousands of billions) feasible configurations.

For all the runs we forced the initial population to be comprised of only feasible individuals and the offspring populations to have at least 80% feasible individuals.

6.2.1.3 Running With an Initial Population

FADSE was started with some good configurations inserted in the initial population. The relaxed borders presented in section 6.2.1.2 are used in this experiment. The main idea is to start from a better point in space hoping that better configurations could be reached in the same amount of generations, or better results could be reached faster.

We selected the optimal configurations, found during the manual exploration (see 6.2.1.1), and inserted them in the initial population. From Figure 6.3-1 and Figure 6.2-2 we concluded that the best configuration in terms of CPI is 2UL2_DL1, the best configuration in terms of energy consumption in 1/4UL2_1/8DL1. We also selected other two configurations which are optimal from both CPI and energy viewpoints: 1/2UL2_1/2DL1 and 1/2UL2_1/4DL1. The vicinities of these four configurations were inserted. The vicinities were obtained by varying the SLVP size, L1 data cache size and L2 unified cache size one step up and down.

FADSE was started again with our 24 selected configurations: the “optimal” manual configurations and their vicinities (some of them are overlapped). The rest of the population (up to 100 individuals) was filled with random individuals.

6.2.1.4 Running With Fuzzy Rules

For this set of experiments, we developed some fuzzy rules derived from our experience in computer architecture design and started FADSE with them. We tested both mutation operators (constant/Gaussian probability to apply fuzzy information) implemented in FADSE when fuzzy information is available. FADSE was started from a random population and with the same relaxed borders used in previous chapters. The rules used in the experiments are:

IF Number_Of_Physical_Register_Sets IS *small/ big* THEN Decode/ Issue/ Commit_Width IS *small/ big*
IF SLVP_size IS *small/ big* THEN L1_Data_Cache IS *big/ small*

The rules are presented here in a simplified form. In the FCL file they are described using 14 rules. We used 2 membership functions for each parameter, one associated to the linguistic term *small*, the other one to *big*.

Virtual parameters were used to describe the L1 data cache (see Chapter 4.3.2.3).

The Mamdani inference system was used (as described in 4.3.1).

6.2.2 Results

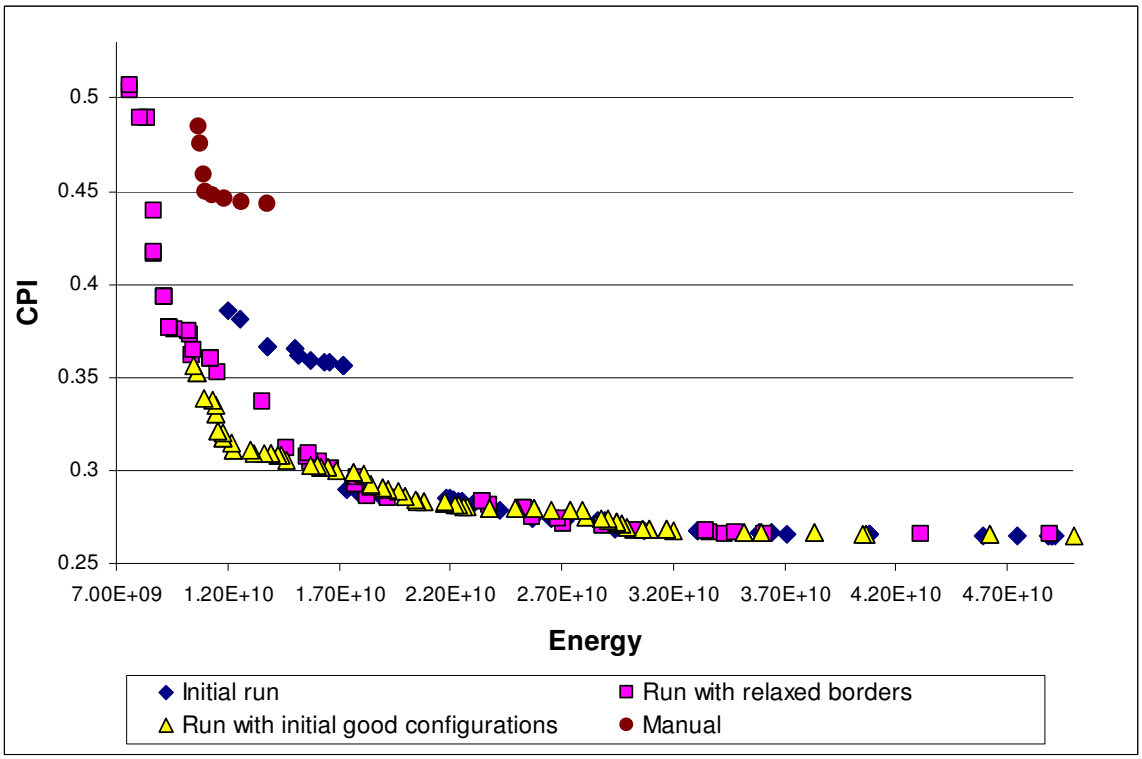

Figure 6.2-3 Pareto fronts comparison of the first runs

In Figure 6.2-3 the first runs are compared: initial run, the run with relaxed borders and run with good individuals inserted in the first population. In terms of CPI all the runs obtain better results than the manual run. From an energy point of view some of the configurations found during the manual exploration are better than the ones obtained during the initial run (before relaxing the constraints – see Section 6.2.1.2).

Relaxing the borders leads to very good results. The found configurations are better on both objectives compared with the manual exploration. The results are evenly distributed along the Pareto front approximation.

The last run depicted in Figure 6.2-3 is the run with initial good configurations. It finds better configurations than the manual exploration and outperforms the initial run, but it is not able to find good configurations from the energy point of view as the run with relaxed borders. It can be observed that the run with initial good configurations finds better results in the in the vicinity of energy 1.20E+10 [$W \cdot cycles$].

The run with initial good configurations does not have such a good distribution of the results (is not able to search the area with low energy) because of the loss in diversity. At the beginning of the algorithm, the initial good configurations are much better than all the random individuals inserted in the population. In the

following generations only these good configurations survive, but they are not very different because they were obtained by varying only 2 from the total of 19 parameters. The mutation operator, with a probability of 0.05 of changing one parameter, is not able to change too much the individuals and, as a result, the algorithm stops in a local minimum. We analyzed the evolution of the Pareto front approximation over the generations to reach this conclusion.

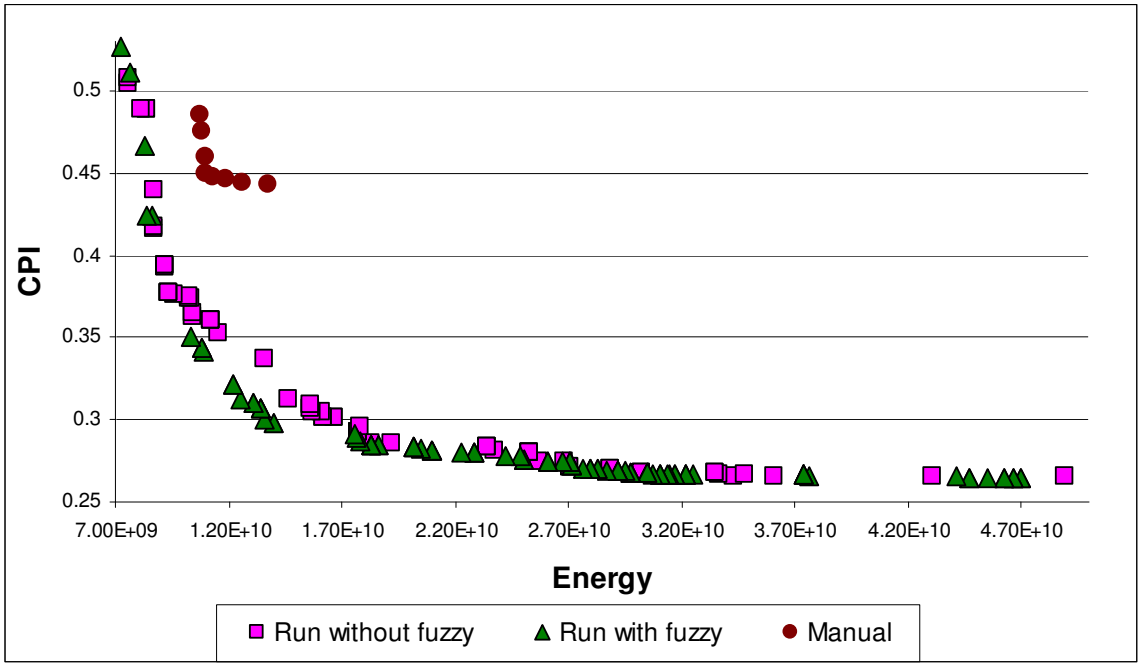

Figure 6.2-4 Pareto front comparisons between the run with fuzzy rules and the run with relaxed borders

From the previous results, we selected the run with relaxed borders as the reference run (called run without fuzzy information in the future) because it obtained the best overall results. It must be noted that all the runs except the first run use the relaxed borders, but are called differently.

In Figure 6.2-4, we compared the run without fuzzy information with the run with fuzzy information (constant probability to apply the fuzzy rules). The run with fuzzy information provides better results especially in the vicinity of energy 1.20E+10 [$W \cdot cycles$].

We also compared the run with fuzzy rules with the one with initial good configurations and we observed that the later obtains a few individuals which are slightly better.

The last Pareto front approximation comparison is made between the two fuzzy runs (see Figure 6.2-5). In this experiment the run with a constant probability to apply the fuzzy rules provides better results, especially in the area with low energies. We can assume that here, like in the run with initial good configurations, there is a loss in diversity. Having a chance of around 80% of applying the rules during the first generations, might lead to very similar individuals. Of course, the situation is not as bad as with the run with initial configurations. In this situation the rules affected only 4 parameters (cache parameters and register file size), as in the run with initial good configuration 16 of them were fixed for the manually inserted individuals.

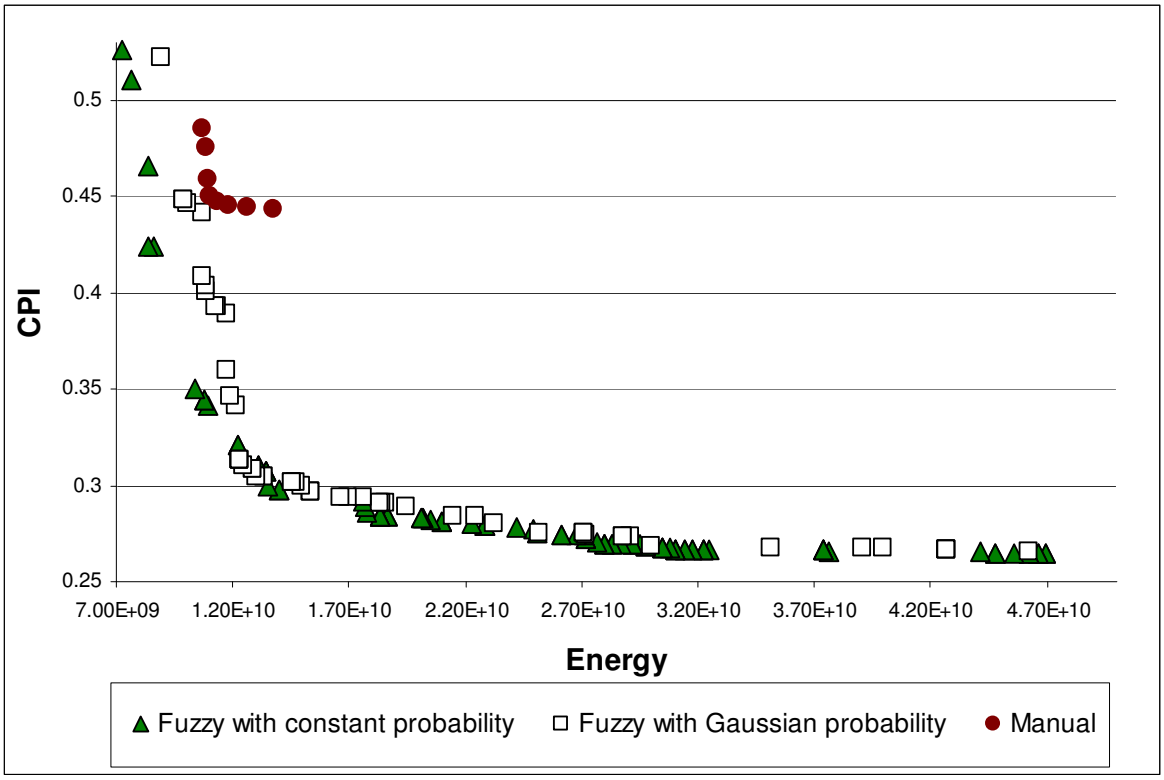

Figure 6.2-5 Pareto fronts comparison between the runs with fuzzy rules

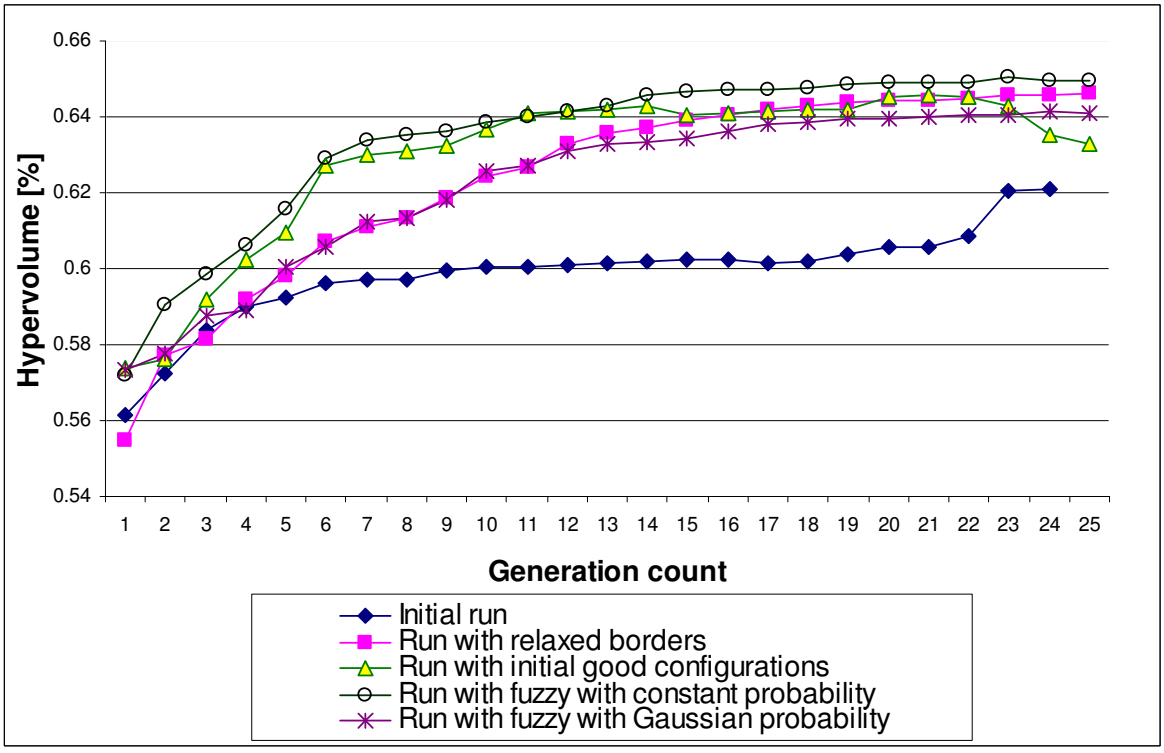

Figure 6.2-6 Hypervolume comparison

As a final comparison, we computed the hypervolume obtained by all the runs during the DSE process (see Figure 6.2-6). This graph gives us information about the convergence of the algorithm and about the quality of results. Except the initial run, the algorithm tends to stop the rapid evolution after generation 15.

After analyzing the evolution of the hypervolume values, we can conclude that:

- The initial run with restricted borders is not able to find so good configurations, the volume covered by the obtained Pareto front approximation is the smallest;
- Running with relaxed borders increased the quality of results considerably even if the design space is larger by a factor of two, from $3.8*10^{13}$ to $7.7*10^{13}$. This is the reason we have decided to use the relaxed borders in all the following experiments presented in this chapter;
- Starting with some initial good configuration can lead to a fast convergence and good results. Still, we observed that the results where grouped only on a part of the objective space, therefore this run falls in a local minimum. This happens because the initial configurations we have provided are very similar. From the 19 parameters we vary, only two of them differ between configurations (number of sets in level 1 and level 2 cache). Problems can be observed at generation 14 and especially after generation 23 where the hypervolume value starts to decline;
- Running with fuzzy rules, with a constant probability to apply them, leads to the best results, both in terms of convergence speed and quality of results;
- Switching to a Gaussian distribution of probability to apply the fuzzy rules leads to worse results. We concluded that the high probability to apply the fuzzy rules lead to a loss in diversity because we have only a few rules with only two membership intervals associated. This means that all the individuals will have the affected parameters very similar. This behavior is enforced for the first generation and the fall back mutation operator is not able to maintain the diversity. In Paragraph 5.5, we also used both methods to compute the probability to apply the fuzzy rules. In that case, better results were obtained with the Gaussian probability. We can explain this through the fact that there were many rules affecting many parameters and with many membership intervals (associated linguistic terms). So, even if the rules pushed the configurations in certain areas of the design space, the diversity of the rules meant that the configurations resulted after the transformation were different.

If we look at the value of the hypervolume at the first generation, for all the algorithms we can conclude that some extra knowledge makes the algorithm start from a better initial population (obvious for the run with initial configurations). Even from this, the algorithms do converge faster.

In terms of time reduction, when using fuzzy rules, we can observe that the hypervolume, obtained by the run with extra knowledge at generation 15, is reached by the run with relaxed borders only at generation 24. Running a generation takes around 24 hours on 96 cores belonging to an Intel Xeon powered HPC system, running at 2GHz. This means that the time required to reach the same quality of results is obtained 9 days earlier. Running with fuzzy rules thus lead to the same quality of results 36% faster, this is a great improvement over the base algorithm. Even more, the hypervolume reached by the run with fuzzy rules is never reached by the other runs during the 25 generations.

To reach generation 25 all the runs evaluate around 2200 individuals. This means a reuse factor of 12% which translates in time reduction for the DSE process (almost three days faster than without a database for 25 generations). The reuse is

quite low compared with what we have obtained in Chapter 5 where a reuse of over 60% has been obtained. The lower reuse factor is determined by the huge design space which makes the algorithms capable of generating new individuals at each generation.

6.2.3 Conclusions

From the manual exploration, we can conclude that the integration of the SLVP in the architecture lead to a better energy consumption at the same CPI because the cache sizes can be reduced. We analyzed the results obtained by the automatic design space exploration process and drawn the same conclusion: with an integrated SLVP the caches can be smaller than in the baseline architecture.

FADSE was able to find good configurations, even if the number of simulated individuals was 2200 from a total of 77 thousands of billions constrained design space (about $3*10^{-11}$%), better than the ones obtained by Árpád Gellért.

Adding extra knowledge improves the results: it helps the algorithm to find better results and also to find them faster. Starting from an initial population with known good configurations, leads to a higher convergence speed. Here some problems were encountered: if the initial configurations are not diverse enough the algorithm might eventually fall in a local minimum. Fuzzy information can provide good results if used wisely: if there are few rules available the constant probability should be used. If the rules have many membership intervals then a Gaussian probability to apply them might lead to better results.

6.3 Optimizing M-SIM3 Architecture

Our next objective was to move to a multi-core architecture. We analyzed many simulators (see Paragraph 6.4) that support multi-core modeling but we were unable to find one that outputs power consumption or area integration (besides IPC/CPI). Since we are focusing on multi-objective optimization we choose M-SIM 3 as simulator since it is the only one that provides multiple (conflicting) objectives.

The parameters for the M-SIM 3 simulator are the ones presented in Table 6.1-1 and Table 6.1-2. For the multi-core configurations we have two identical cores. As objectives, we try to optimize the same ones as for M-SIM 2 (see Paragraph 6.2): energy and CPI.

6.3.1 Methodology

The same methodology as for M-SIM2 has been used:

- NSGA-II algorithm;
- Population size 100;
- Bit flip mutation with a probability of 0.05;
- Single point crossover with a probability to apply crossover of 0.9.

First, we ran M-SIM 3 as a single core with the same benchmarks as M-SIM 2. The difference from the previous explorations is that we forced only the first generation to be comprised from feasible individuals. For the rest of the design process we are not forcing any number of feasible individuals (individuals that respect all the constraints) in the offspring population.

The next step was to move to a multi-core architecture. We paired the benchmarks used in the previous section in the following way: {twolf, vpr}, {applu, equake}, {bzip2, gcc}, {galgel, lucas}, {gzip, mcf}, {mesa, mgrid}. The benchmarks were selected as in [39] with the exception that *parser* is replaced with *mcf* in our

work due to some incompatibilities with the HPC system used during simulation. The SLVP implementation has not been used because it is not integrated into M-SIM 3 yet.

The connector implemented for M-SIM 3 (with the help of student Camil Băncioiu) can be configured to accept homogeneous and heterogeneous configurations. For a homogeneous multi-core, the user has to specify a single set of parameters and then the connector applies the same values for all the cores. If a heterogeneous architecture is required, the user has to specify all the parameters in the input XML file and they will be varied by FADSE. Since the design space is huge and running a single generation took more than 36 hours, we decided to run this experiment while simulating a homogeneous multi-core**,** to reduce the number of possible configurations.

6.3.2 Results

In our first test we ran with M-SIM 3 as single core. In Figure 6.3-1 the evolution of the hypervolume can be seen. The shape obtained is slightly different from what we have obtained in all our previous explorations. For the first 5 generations the evolution of the hypervolume is not typical. We examined the individuals from the offspring population and concluded that over 60% of them are infeasible. This percentage decreases to 50% at the last generation, but still many are infeasible. This means that the number of offspring, from which the algorithm can choose good individuals, is not very high, which translates in a lower convergence speed. The time, required to evaluate an entire generation, is also reduced because the individuals are tested if they respect the rules before sending them to evaluation. The small number of feasible individuals explains the results from Figure 6.3-2. The number of accepted individuals in the next population is depicted against the number of new individuals generated (infeasible included). We can observe that the number of accepted individuals is not larger than 30, even at the second generation (the first generation is always 100 since all the individuals form the next parent population).

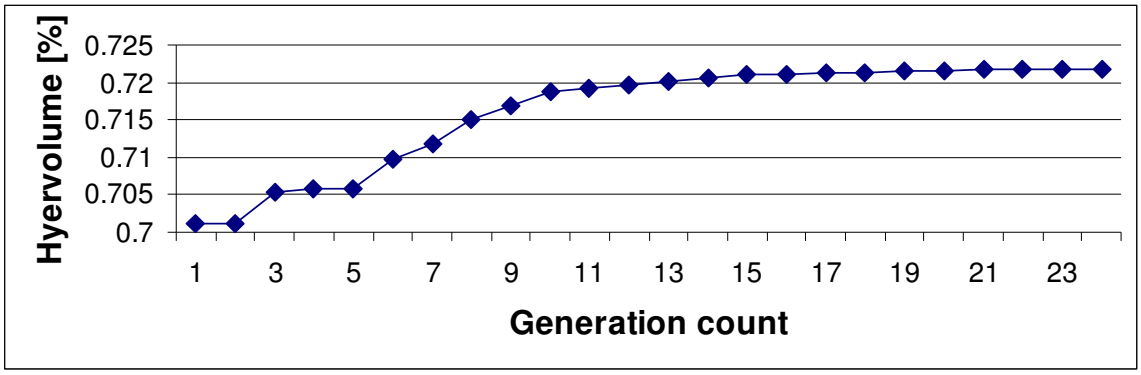

Figure 6.3-1 Hypervolume obtained for the M-SIM 3 configured as a mono-core

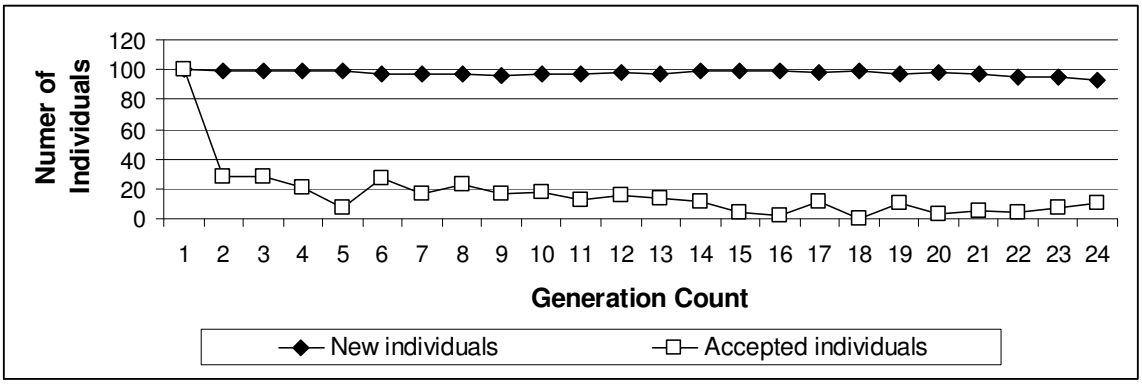

Figure 6.3-2 Number of unique offspring individuals versus the number of accepted individuals in the next population – mono-core run

This run with a mono-core processor on M-SIM 3 was our first experiment with this simulator and with constraints in general. It determined us to implement a technique to force a certain percentage of feasible individuals in the offspring population.

The next experiment presented in this chapter is the exploration of a dual-core architecture. In this experiment, we forced the minimum feasible number of individuals in the offspring population to 80%. The evolution of the hypervolume presented in Figure 6.3-3 shows the convergence of the algorithm. Compared to Figure 6.3-1, the evolution is more smoothly. A difference from previous exploration can be seen when comparing Figure 6.3-2 with Figure 6.3-4. In Figure 6.3-4 around 60 individuals are accepted in the next population during the first generation. Only after 13 generations the number decreases to the number of individuals accepted in the previous exploration since the second generation.

In this exploration we observed an 18% reuse. The lower reuse is explained by the very large design space. It is visible that in Figure 6.3-4 the algorithm produces almost 80 new individuals even at the last generations.

With this last exploration we proved that FADSE can be used with multi-core simulators and can run for extended periods of time (over a month).

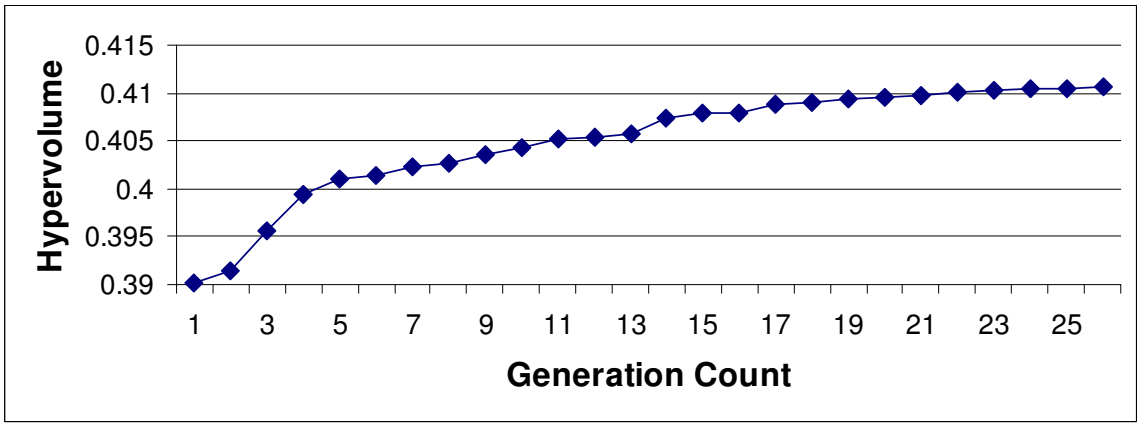

Figure 6.3-3 Hypervolume obtained for the M-SIM3 configured as a multi-core

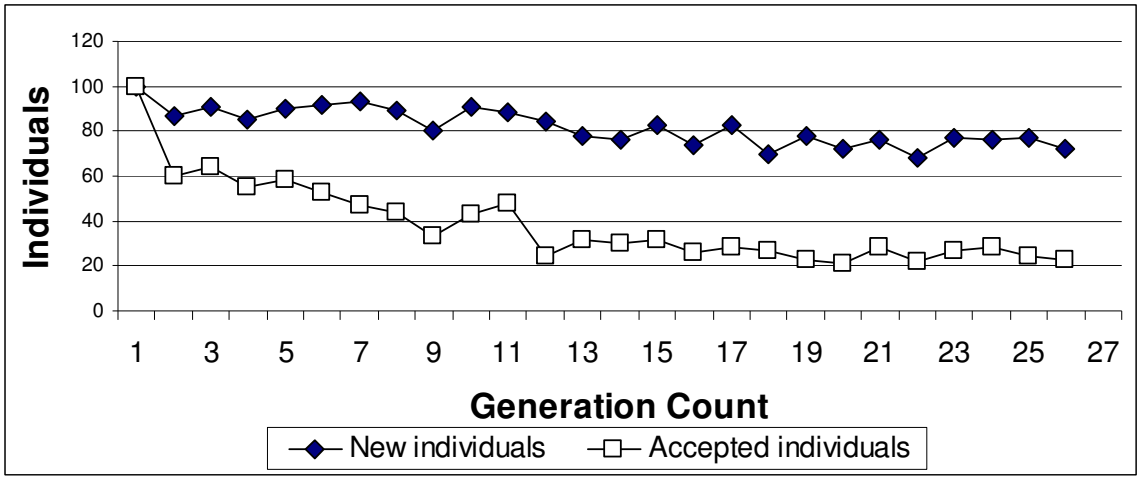

Figure 6.3-4 Number of unique offspring individuals versus the number of accepted individuals in the next population – multi-core run

6.4 Multi-core Simulators Considered for Optimization

Multiple multi-core simulators were considered for an optimization using FADSE. We have analyzed all these simulators to try to find their strong points and their weaknesses. An overview is done in this chapter.

6.4.1 UNIted SIMulation Environment

UNIted SIMulation environment (UNISIM) [46] was the first multi-core simulator we used. The UNISIM simulator contains two major parts: a cycle-by-cycle simulator and a transaction level modeling (TLM) simulator.

The cycle-by-cycle simulator models a multi-core 32 bit PowerPC 405 RISC architecture with a 5 stage pipeline and separated data and instruction caches (Harvard architecture). The cycle-by-cycle simulator could not be configured using a command line. To influence the parameters (cache size, number of CPUs, etc.) the code had to be changed. We have developed our custom tool that could receive parameters from an external source generate the required code automatically, inject it into the simulator source code, compile the simulator and then run the configuration with the specified benchmark.

To run parallel benchmarks UNISIM supports the POSIX Pthread library. With this we were able to write our own test programs. We implemented different sort algorithms (quick sort, merge sort) and matrix multiplication methods in a parallel fashion. Given that, we implemented new coherency protocols (MSI), besides the already existent ones (MESI), we also needed some applications to test the correctitude of the implementation. For this, we developed a test application that was preserving the order of operations in mathematical computation using threads: the adding thread had to wait for the multiplying thread, etc.

The TLM simulator provided a very fast simulation. On this version we were able to run benchmarks from the PARSEC [47] and SPLASH-2 [48] suites.

We did not use UNISIM since it does not provide any information about the power consumption or area integration. Integrating such a functionality proved to be difficult.

6.4.2 M5

M5 [49] is a well known full system multi-core simulator. It was recently merged with the GEMS simulator and called gem5.

M5 is able to simulate Alpha architectures. Supported ISAs are: Alpha, MIPS, Sparc. It can run in two modes: full system and application only. In full system it is able to boot the Linux operating system. It supports different levels of detailed simulation: from very fast inaccurate simulation to very accurate (and slow). These levels of detail can be changed during a simulation process so, for example, the boot of the Linux kernel can be simulated with low accuracy but when the benchmark is started the accuracy is increased.

M5 can simulate multi-computer systems, tied through networks. We were able to run the SPLASH-2 suite of benchmarks in the application only mode. On the full system simulator we also ran the SPLASH-2 benchmarks and our own applications.

M5 lacks the support for power consumption estimation or other objectives besides the speed related ones.

A FADSE connector for M5 has been developed.

6.4.3 Multi2Sim

Multi2Sim simulates a multi-core multi-threaded superscalar pipelined architecture. It supports the x86 ISA which allows it to run benchmarks compiled. It is an application only simulator (no full system) and supports most of the benchmarks suites available: SPLASH2, PARSEC, SPEC, etc. It is also able to simulate OpenCL programs.

We integrated Multi2Sim with FADSE using a connector.

Multi2Sim does not include any power or area of integration outputs but the authors provide a list of outputs that can be used in conjunction with McPAT [50] to extract these metrics. McPAT is a library which is able to provide information about power consumption, area integration for multi-core or System on Chip (SoC) architectures. It is easily configured through a XML interface. We developed a tool that can extract metrics from the outputs of the Multi2Sim and insert them automatically in the XML configuration file of McPAT. The problem is that Multi2Sim outputs only dynamic information about the accesses in caches, memory, and not about the actual structure of system architecture. This meant that we had to use the default Alpha architecture implemented by McPAT as a base architecture and insert only the number of accesses. This gave us an estimation of the dynamic power consumed by the simulated architecture, but not more. Our script is limited, for the time being, at single-core architectures, but a multi-core implementation could be easily provided.

6.4.4 SuperESCalar Simulator

SuperESCalar Simulator (SESC) [51] multi-core capable cycle level accurate simulator developed at the University of California. The advantage of this simulator is that it integrates with HotSpot [52], Wattch [43] and Cacti [53]. Through this tools it can provide to the user (besides the usual CPI) the power consumption and area required by the configuration.

The RedHat Linux based HPC computer from “Lucian Blaga” University of Sibiu, on which we ran the simulations, was not supported by the SESC simulator. Because of this, the project was abandoned and no connector was written for it.

6.4.5 SCoPE

SCoPE is a simulator used together with M3Explorer. A connector has been developed by the authors of SCoPE and M3Explorer for the M3Explorer DSE tool. Since FADSE is similar at the interface level with M3Explorer, a FADSE connector for the SCoPE simulator would require very little time to develop. Since SCoPE can simulate different architectures from multi-cores to multi-processor SoCs and it outputs performance metrics and power consumption it was a good candidate for a DSE with FADSE. Another advantage is that SCoPE is a TLM simulator meaning that it simulates extremely fast, a good feature for the lengthy design space explorations.

Together with the student Camil Bancioiu we tried to use this simulator in our work. We were able to describe our own extensible multi-core architecture based on ARM architecture. On this architecture, we ran the single-threaded MPEG benchmark. When we extended the work to multi-core architectures we encountered errors in the code. At a closer analysis of the code, we observed that the current version of the simulator did not support data caches, a feature very important in our opinion. We contacted the authors of the SCoPE simulator and a new version is in development that will solve these problems.

6.4.6 Graphite

Graphite is a simulator for multi-core architectures. The novelty of this simulator compared to the previous analyzed ones is the distributed simulation. Its scope is to allow the exploration of systems with dozens to thousands of cores. The simulator creates a thread for each core in the simulator. This pool of threads is then distributed on all the cores from all the hosts in the network.

We are currently working in integrating this simulator with FADSE.

"An ounce of action is worth a ton of theory."

Friedrich Engels

7 Multi-objective Optimization of System on Chip Architectures

This chapter presents our proposed method for performing an automatic application driven design space exploration for System-on-Chip (SoC) designs. We connect FADSE with UniMap [54], a SoC simulator. The goal is that, for a given application, to automatically find the best SoC architecture, in a multi-objective way. We have three objectives: SoC energy consumption, SoC area and application runtime.

With UniMap, we model and simulate an entire System-on-Chip, made of tens of heterogeneous Intellectual Property (IP) cores, mapped onto the tiles of a Network-on-Chip interconnection network.

We propose a practical DSE workflow, which allows us to determine, for any particular application, the SoC designs that consume the smallest amount of energy, occupy the smallest area and allow the application to execute the fastest. The DSE process is performed with multi-objective algorithms from two classes. Having two genetics and two bio-inspired algorithms, we compare four multi-objective techniques aiming to find the algorithm that performs the best.

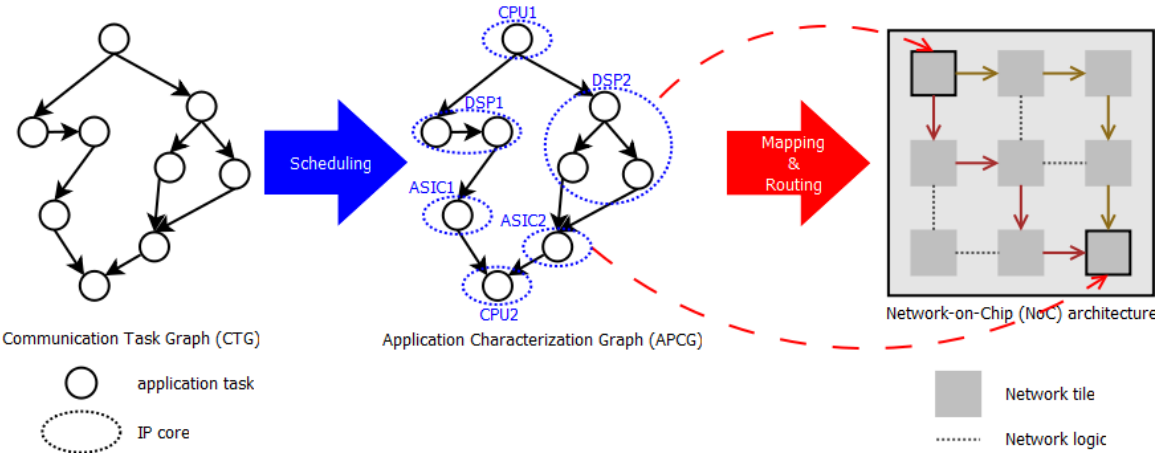

Figure 6.4-1 The scheduling, application mapping and routing problems [55]

The APCG is used as input for an application mapping algorithm, which maps the given IP cores onto the nodes of a given NoC architecture, such that different metrics of interest are optimized. Obviously, the mapping algorithm must be aware of the routing algorithm, i.e. how data is sent from one network node to another.

7.1 UniMap Overview

UniMap integrates different mapping algorithms and also a Network-on-Chip simulator. Some algorithms are available in literature and some were improved by the author. The NoC simulator is also developed by Ciprian Radu. UniMap was developed so that different algorithms may be evaluated and optimized in a unified manner, on multiple NoC designs. If a NoC architecture is given, UniMap can be used to find the best mapping in terms of energy consumption, network latency, etc. for any parallel application.

7.2 Design Space Exploration Workflow

Ideally would be to search for the best NoC architecture for every possible application mapping. This is actually the exhaustive approach in which we would take every possible placement of IP cores onto the NoC nodes and for each placement we would try all the available NoC designs and evaluate their performance using ns-3 NoC, UniMap's simulator. The DSE mechanism described is practically a DSE in an inner DSE because the UniMap DSE includes FADSE. Our DSE workflow can be made faster by serializing FADSE DSE after UniMap DSE. By doing so, we evaluate each mapping on just one SoC design. UniMap will output a Pareto front and FADSE will take the mappings from this front and search for each one the best System-on-Chip.

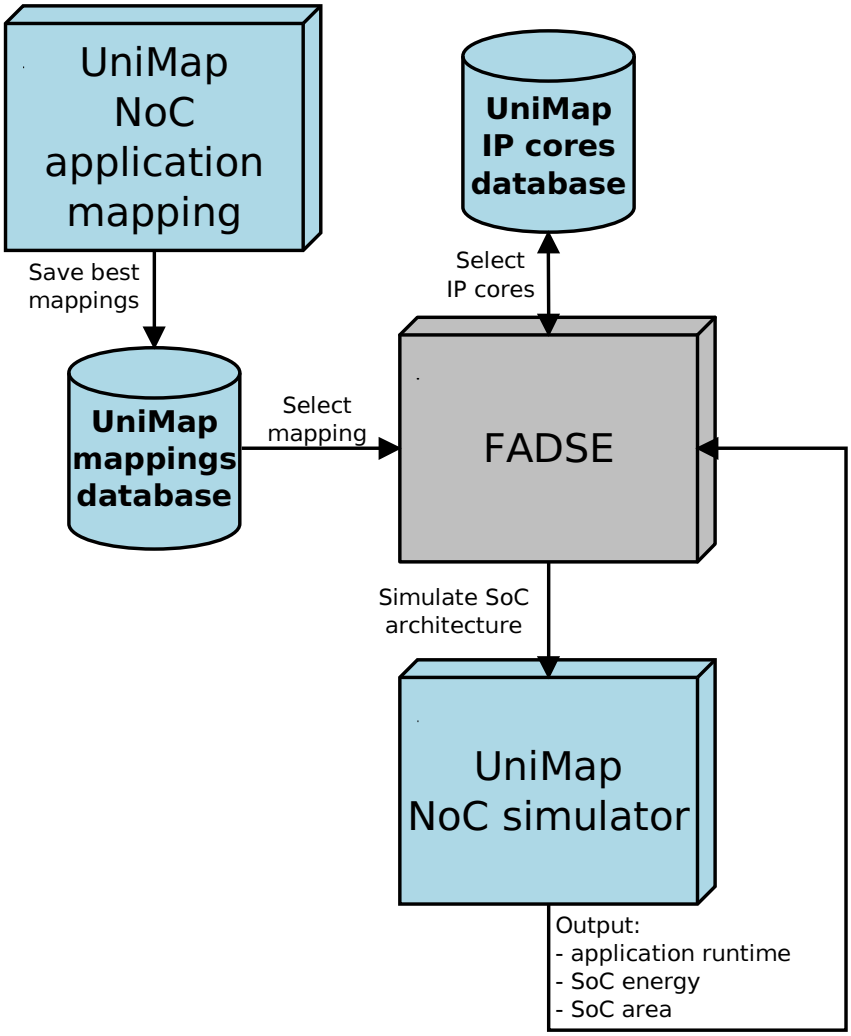

Figure 7.2-1 Application driven DSE workflow for SoC designs

We can still reduce the time complexity of our DSE approach by letting UniMap use an analytical model for evaluating the application mappings, on a default System-on-Chip design. UniMap no longer uses a NoC simulator to evaluate mappings. This DSE workflow is less accurate than the previous two, but it is more feasible. Using an analytical model, we can evaluate a mapping in less than a second. With a bit energy model we estimated the NoC communication energy on our HPC system.

Our Design Space Exploration workflow begins from UniMap, which maps applications onto Network-on-Chip designs. Each mapping is evaluated with an analytical model [56] that estimates the NoC communication energy. For every application, UniMap saves the best mappings it finds into a database.

FADSE then searches, for each application, the best System-on-Chip architecture. The first ten best mappings[1] found with UniMap are used by FADSE (for each application). These best mappings are taken from all best mappings obtained with all UniMap heuristic algorithms: Simulated Annealing, Branch and Bound, Optimized Simulated Annealing and Elitist Genetic Algorithm and Elitist Evolutionary Strategy, with all their variants [55].

Then we use FADSE to do a Design Space Exploration, guided by a multi-objective algorithm. Different System-on-Chip designs are simulated by FADSE. The mapping provided to FADSE says where each IP core is placed onto the NoC. It also says what type of IP core is associated to each task. However, FADSE will automatically simulate with other compatible IP core types as well. After it chooses the core types, FADSE generates a System-on-Chip by topologically placing the cores onto the NoC. FADSE automatically configures the Network-on-Chip. After that, UniMap's ns-3 NoC simulator is called by FADSE. This simulator measures application runtime, SoC energy and SoC area (our three DSE objectives).

We work with the E3S [57] IP core library. It provides information regarding the power consumed by each IP core for running a particular application task. IP core area and power consumption, while the core is idle, are also specified.

Our UniMap NoC simulator integrates ORION 2.0 [58]. This allows us to estimate Network-on-Chip power and area. We consider both leakage and dynamic power, for routers and communication links. In a similar way, the area occupied by the NoC is computed as the sum of routers' and links' area.

Each application runs for a given number of CTG iterations. Application runtime is the simulation time required by the benchmark to finish. A CTG iteration means running a benchmark until it is completed. Several iterations mean that we restart the benchmark after a specified (in the benchmark description) amount of time. An example can be for MPEG where a new frame has to be processed every 1/24 seconds. This might lead to congestions in the network. The number of CTG iterations is determined empirically such that the simulations run fast enough that our DSE finishes in a reasonable amount of time.

Our proposed DSE workflow outputs a Pareto front with the near optimal System-on-Chip architectures found for a given application.

The following section describes our experimental methodology. We give details regarding how exactly we did the simulations, what benchmarks we used, the varied design parameters and how we configured UniMap and FADSE. During our DSE process we did not modify the Network-on-Chip topology. This is the NoC element that is essentially used by the mapping algorithms. Modifying it would create inconsistencies. Network-on-Chip application mapping is by definition topology dependent. Nevertheless, our workflow might be applied to different NoC topologies. This would have the advantage of finding the most suitable NoC topology, as well. In order to do so, we would need to adapt our mapping algorithms for these other Network-on-Chip topologies as well.

[1] Depending on resources available, a bigger number of best mappings may be used

7.3 Methodology

Since we present in this thesis only preliminary results, we worked with just with four benchmarks: telecom, MPEG-4, H.264 (CTG 0) and VOPD (CTG 0). We plan to do more simulations in the future and to use more applications.

Using all application mapping algorithms from UniMap, we selected the first ten best mappings that we found for each application mentioned above. telecom is a benchmark with 30 IP cores, which are mapped on a 6x5 2D mesh Network-on-Chip. MPEG-4 has 12 IP cores and H.264 (CTG 0) and VOPD have 16. MPEG-4 is mapped onto a 4x3 2D mesh and H.264 (CTG 0) and VOPD use a 4x4 2D mesh.

A Communication Task Graph (CTG) describes an application though its traffic pattern. The CTG can be reiterated multiple times. By doing so, we can simulate successive executions of the application. Since we only afforded to run each benchmark for less than ten minutes, we chosed the number of CTG iterations accordingly. The following table illustrates this aspect.

Benchmark	CTG iterations
telecom	10
MPEG-4	2
H.264 (CTG 0)	4
VOPD (CTG 0)	1

We worked with the IP core library provided by E3S. It contains a total of 34 cores. telecom is an E3S benchmark. As such, we know exactly what cores can execute each of its tasks. There are on average 20 core types for each task of the telecom application. For the other three benchmarks, which are not from the E3S suite, we consider that any IP core from the E3S library can execute any task of our non E3S benchmarks. This is possible because E3S considers for each IP core a generic task, for which we know the execution time and power consumption.

We have considered the following Network-on-Chip parameters: network clock frequency, input buffer size, flit size, packet size and routing protocol. The NoCs frequency is varied from 100 MHz to 1 GHz, using a step of 100 MHz. The input buffers can uniformly hold from one to ten flits. The flit size is in bytes, starting from 4 and going up to 256, using a geometric progression with ratio two. The packet size is minimum two flits and maximum ten flits. The routing protocol can be either XY or YX (both are variants of Dimension Order Routing). We automatically set the NoC bandwidth to a value that allows one flit to be transmitted in one Network-on-Chip clock cycle.

The next table presents the size of the search space for each benchmark. We have a maximum of $N = 12600$ possible NoC designs but, the search space generated by the IP core types (C) is considerably much bigger.

Benchmark	Search space size
telecom	$12600 \cdot 20^{30} \approx 1.35 \cdot 10^{43}$
MPEG-4	$12600 \cdot 34^{12} \approx 3 \cdot 10^{22}$
H.264 (CTG 0)	$12600 \cdot 34^{16} \approx 4 \cdot 10^{28}$
VOPD (CTG 0)	$12600 \cdot 34^{16} \approx 4 \cdot 10^{28}$

We configured our Framework for Automatic Design Space Exploration to work for this research with four multi-objective techniques: NSGA-II, SPEA2, SMPSO and OMPSO. We set all algorithms to stop after 50 generations.

NSGA-II was set to work with a population of 100 individuals. Single point crossover, bit flip mutation and binary tournament selection were the well known genetic operators that we used. The benchmark telecom has the highest number of IP cores: 30. We also vary with FADSE five NoC parameters. Thus, our largest chromosome has 35 genes. This suggests a mutation probability of 3% (*1/n*, where *n* is the number of parameters). The crossover probability was set to 90%.

SPEA2 was configured exactly like NSGA-II and the archive size was set to 100.

In the same manner, SMPSO and OMOPSO have an archive of size 100 and they work with a swarm of 100 particles.

7.4 Results

In this paragraph we present the results obtained using the DSE technique described in the previous paragraph. Due to time required for simulation we were able to explore ten mappings only on the telecom benchmark. On the other benchmarks we were able to explore a single mapping: the best one found analytically.

7.4.1 Design Space Exploration on the Telecom Benchmark

We ran telecom on the first ten best mappings and computed the hypervolume for each one and then we averaged the results. The results are presented in Figure 7.4-1. From this metric we can conclude that the particle swarm optimization algorithms (OMOPSO and SMPSO) converge faster than the genetic ones (NSGA-II and SPEA2), but the results obtained after 9-10 generations by the genetic algorithm are better from the quality point of view. It is interesting to observe that the algorithms from the same class (genetic/PSO) obtain similar results. It is more important the type of the algorithm than the actual specific implementation. The two genetic algorithms have similar results with a bit faster convergence for the NSGA-II algorithm, but the final hypervolume value is slightly better for SPEA2. SMPSO has better results than OMOPSO from both convergence speed and quality of results point of view.

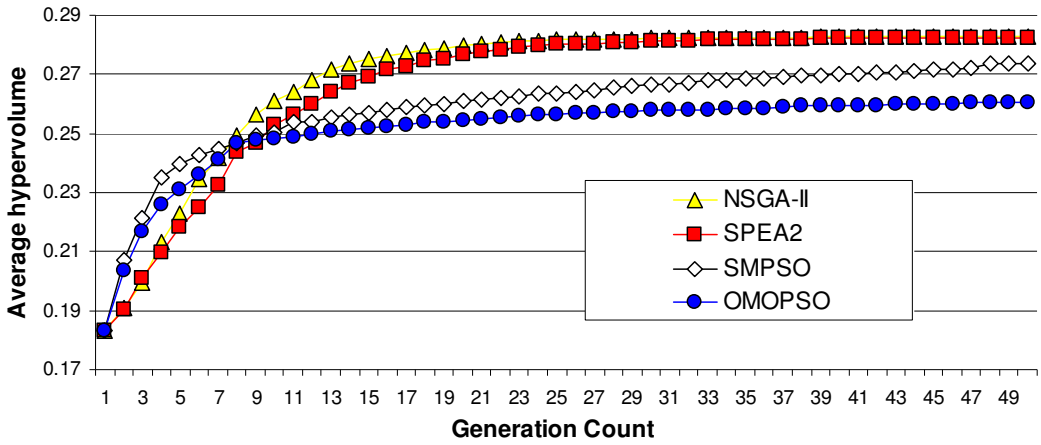

Figure 7.4-1 Average hypervolume over all ten best telecom mappings

The next step was to use the coverage metric to compare the results obtained by the algorithms. We performed comparisons between all the algorithms on all the

benchmarks. Due to space constraints we present only the comparison performed between a genetic algorithm (SPEA2) and a particle swarm optimization (OMOPSO). The results are depicted in Figure 7.4-2.

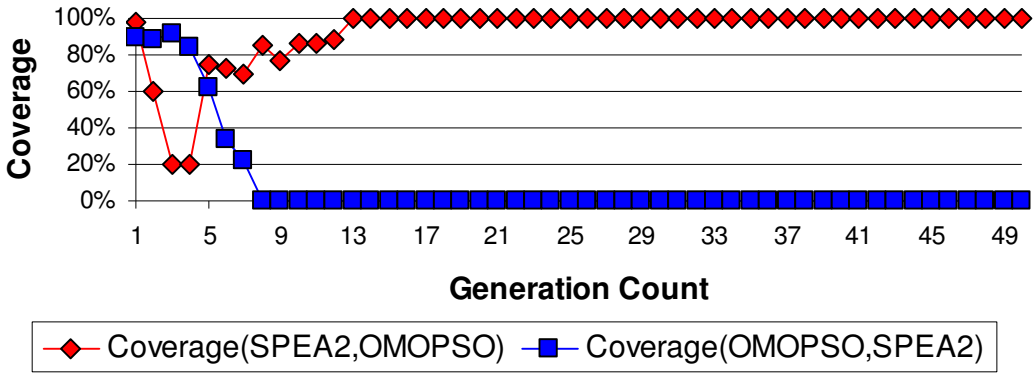

Figure 7.4-2 Coverage comparison between SPEA2 and SMPSO on the telecom benchmark

It can be seen that the results obtained by the genetic algorithm clearly dominate the results obtained by the PSO algorithm. From the other comparisons using the coverage metric (not shown here) we observed that SPEA2 is better than NSGA-II and that from the PSO algorithms OMOPSO has a better coverage (different than the information given by the hypervolume).

Since the hypervolume metric is mostly used for measuring convergence speed and we proved in previous experiments that coverage can be misleading, we decided to compare the obtained Pareto fronts approximation. The results are shown in Figure 7.4-3. We took the best results found by all the algorithms and kept only the nondominated points. We observed that the nondominated points are only obtained by the genetic algorithms. Another observation is that results are found from all the ten mappings, even if, by using the analytical model, mapping 1 is the best one (only from the energy point of view). We analyzed the results considering the last observation and we saw that the best energy is obtained for mapping 6. The architecture with the smallest area was found for mappings 3 and 5. The two mappings are nondominated as mapping 3 has better energy consumption while mapping 5 has a better application runtime. From the application runtime point of view again mapping 6 is the best one.

We would have expected to obtain the best results, at least from an energy point of view with the first mapping. The results can be explained by the fact that the analytical model does not take into consideration the eventual collisions (network congestion) that might appear in the network while the simulator can take them into account. Another reason for this result is the fact that FADSE does not exhaustively explore the design space. This means that for a certain mapping we might find a hardware configuration which is very good but was not discovered in the other explorations. These results will be further explored and analyzed in future work.

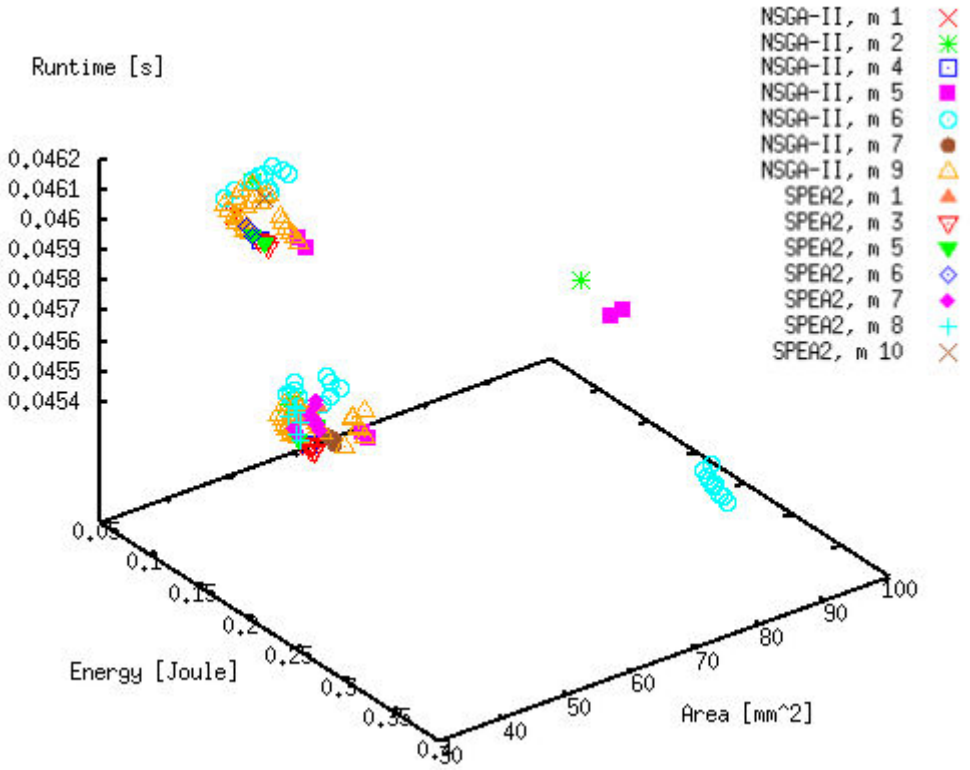

Figure 7.4-3 Best configurations found by all four algorithms for telecom benchmark

We analyzed the best SoC configurations found for each mapping for the telecom benchmark. As we said, for mapping 6 we found the best architecture from both energy and runtime point of view. The two SoC designs using this mapping were found by the NSGA-II algorithm. SPEA2 found the best configurations from an area point of view. The following table describes the architectural parameters of the SoC designs. The table is also presented by us in [55].

Objective	Algorithm	Mapping	NoC parameters					SoC energy [Joule]	SoC area [mm^2]	Application runtime [ms]
			Frequency [MHz]	Buffer size [flits]	Flit size [bytes]	Packet size [flits]	Routing			
Energy	NSGA-II	6	100	4	4	10	YX	0.095159	50.113	46.1144
Area	SPEA2	5	200	1	4	10	XY	0.158177	37.366	46.1132
Area	SPEA2	3	400	1	4	10	YX	0.167928	37.366	46.1111
Runtime	NSGA-II	6	900	4	32	6	YX	0.341914	81.227	45.4

As expected, we obtained the lowest energy consumption with the smallest frequency allowed in our DSE process.

In accordance with our intuition, we obtained the lowest energy with a System-on-Chip design that uses a NoC running at the minimum frequency permitted by our DSE model. We also observed that the SoCs having the smallest area use a NoC with buffers of just one flit in size and some of the smallest IP cores. Our two area optimal SoC designs use only a quarter of the NoC buffering resources used by our best energy and application runtime SoC architectures. Our two area-optimal SoCs practically differ by their Network-on-Chip frequency. One of the two SoCs is faster than the other because it uses a NoC twice faster. In terms of application runtime, our optimal runtime SoC is more than half a millisecond faster than our other three optimal Systems-on-Chip. This fastest SoC design uses a much faster Network-

on-Chip and bigger packets. This obviously impacts on SoC energy and area. Finally, we observe our optimal SoCs use both XY and YX routing. This shows that routing algorithms influence the SoC's performance.

7.4.2 Design Space Exploration on the MPEG-4 benchmark

Our next experiment was done on the best mapping analytically found for the MPEG-4 benchmark. Since the time required for simulation on this benchmark was longer, we were able to perform a DSE only on one mapping. We computed all the metrics like in the previous experiment. In Figure 7.4-4 we present the hypervolume value obtained by each algorithm during the 50 generations.

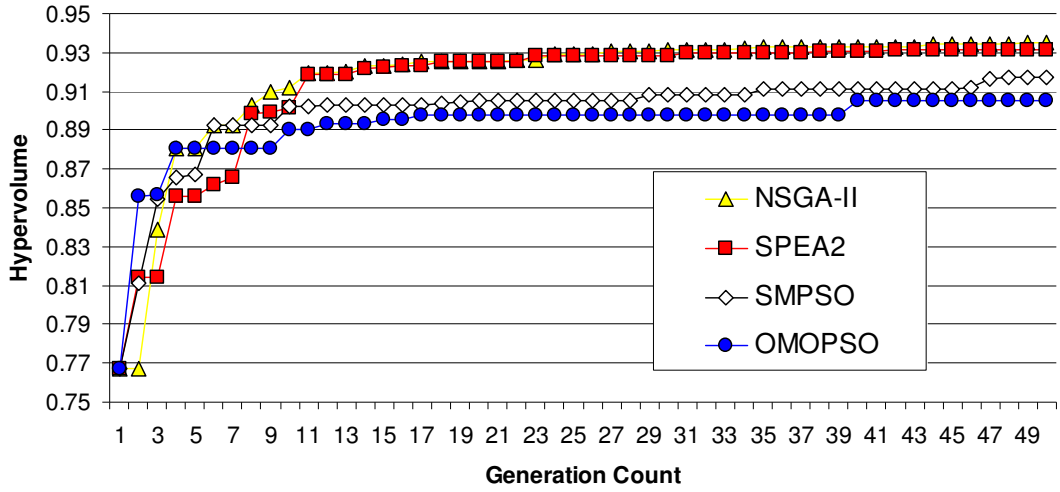

Figure 7.4-4 Hypervolume obtained by all the algorithms for the MPEG-4 benchmark

Since we ran only on one mapping and do not present an average results the curve is not as smooth as in previous experiment but the conclusions remain fairly the same. The two genetic algorithms continue to find the best SoC designs, while the PSO algorithms do not provide such good results. Again the PSO algorithms converge very fast but this time NSGA-II has a very similar behavior. Form this point of view it seems that NSGA-II finds the best results. From the perspective of results the PSO algorithms perform poorly compared to the genetic ones.

Similar with previous chapter, we compute the coverage metric. First, we compare the two genetic algorithms with the purpose of selecting the best one from the coverage metric point of view. The results are shown in Figure 7.4-5. For the first 37 generations the algorithms have similar results. After these generations SPEA2 seems to gain an advantage. This is somehow different from what we concluded from the hypervolume metric. Nevertheless, we decided to choose SPEA2 as the best genetic algorithm in this experiment.

A comparison was made between the Pareto fronts approximation found by the two genetic algorithms. We could not decide on one best algorithm. NSGA-II finds better results in some areas of the space while its solutions are dominated (by SPEA2 solutions) in other regions. Still we did observe a slightly larger spread of solutions found by the NSGA-II algorithm. Choosing a winner is hard and it truly depends on the requirements of the designer.

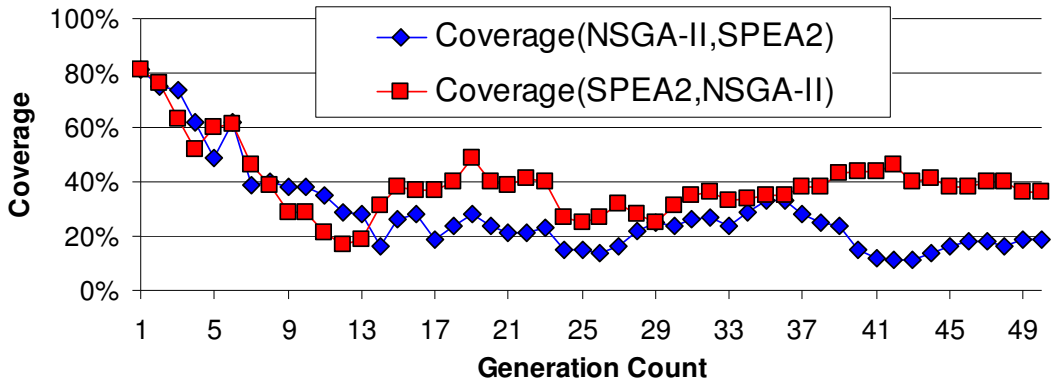

Figure 7.4-5 Coverage comparison between NSGA-II and SPEA2, for MPEG-4

Second in Figure 7.4-6 we perform a comparison between the two selected PSO algorithms: OMOPSO and SMPSO. For the first generations OMOPSO performs significantly better and even after generation 20 it is still better from the coverage point of view. After 45 generations the results are similar. We selected OMOPSO as the best algorithm from the PSO ones because it has an overall better quality of results. As in the previous comparison we looked at the Pareto front approximations obtained by the two PSO algorithms. From our (subjective) perspective the SMPSO algorithm seemed to have better results (better spread of solutions).

For our final comparison using the coverage metric, we selected SPEA2 and OMOPSO, the best performing algorithms from this point of view from the previous comparisons. The results are presented in Figure 7.4-7. Due to the faster convergence of the PSO algorithm during the first generations, the individuals obtained by OMOPSO dominate the individuals obtained by SPEA2. But after 7-8 generations the genetic algorithm manages to surpass the PSO algorithm reaching an almost 100% domination. We analyzed the final Pareto fronts approximations of both algorithms and the conclusion remains the same: the genetic algorithm obtains clearly better results.

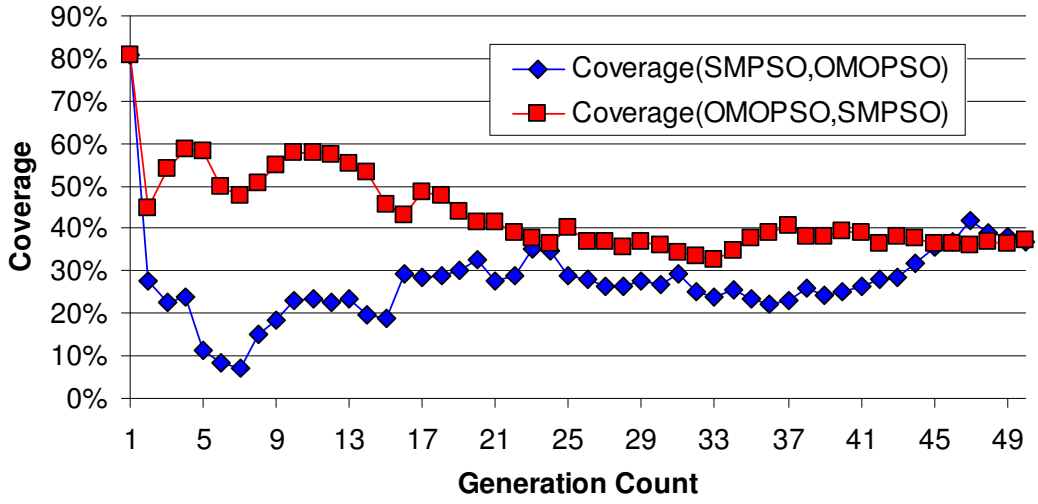

Figure 7.4-6 Coverage comparison between SMPSO and OMOPSO, for MPEG-4

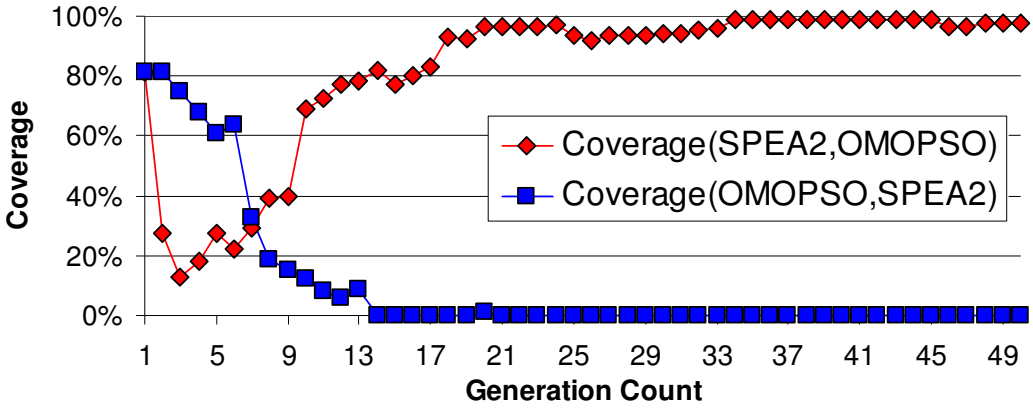

Figure 7.4-7 Coverage comparison between SPEA2 and OMOPSO, for MPEG-4

As a final result in this experiment we present one of the Pareto fronts approximations found by the algorithms. We selected the front obtained by NSGA-II since from our perspective it had the best results: the best spread of solutions, a good hypervolume value and even from a coverage point of view quite close to the SPEA2 algorithm. The Pareto front approximation is depicted in Figure 7.4-8 (since this is a joint work, we presented this figure also in [55]). The figure depicts an interpolation of the points for a better visibility of the obtained 3D surface.

With this we proved that FADSE is able to find good configurations on problems with three objectives and that the obtained solutions are spread along a surface and here is no single solution that is the best on all the objectives (the objectives are contradictory).

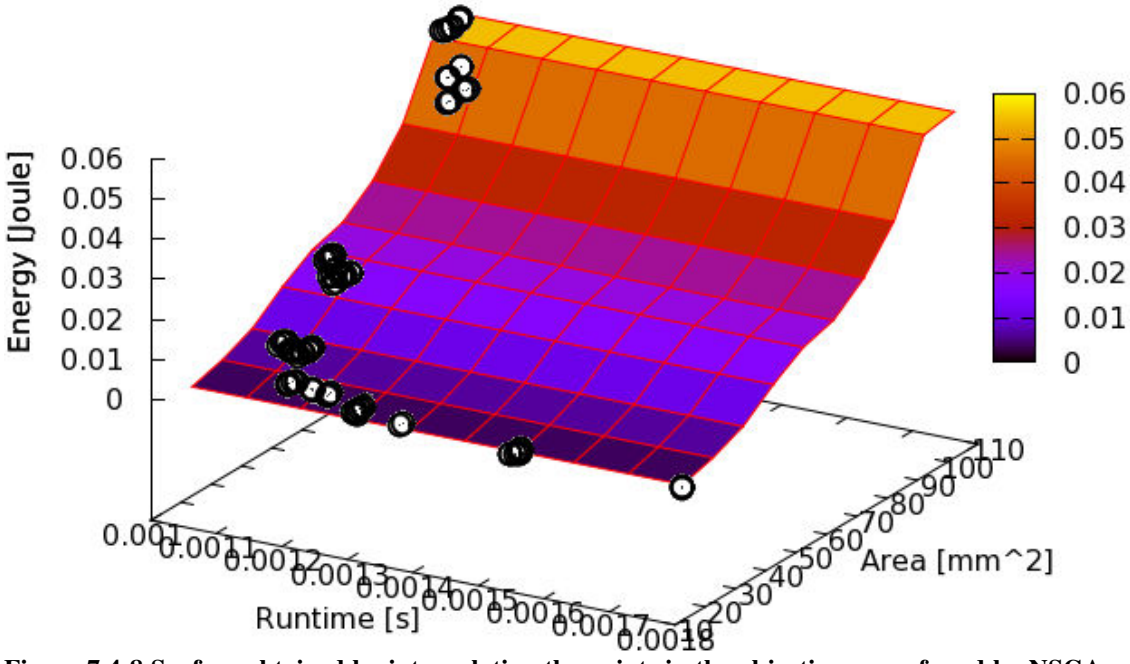

Figure 7.4-8 Surface obtained by interpolating the points in the objective space found by NSGA-II

7.4.3 Design Space Exploration on H.264 and VOPD benchmarks

The last experiments were conducted on H.264 and VOPD benchmarks. The obtained results were similar with the ones from the previous experiments. Due to space constrains we present only the hypervolume values obtained; see Figure 7.4-9 and Figure 7.4-10.

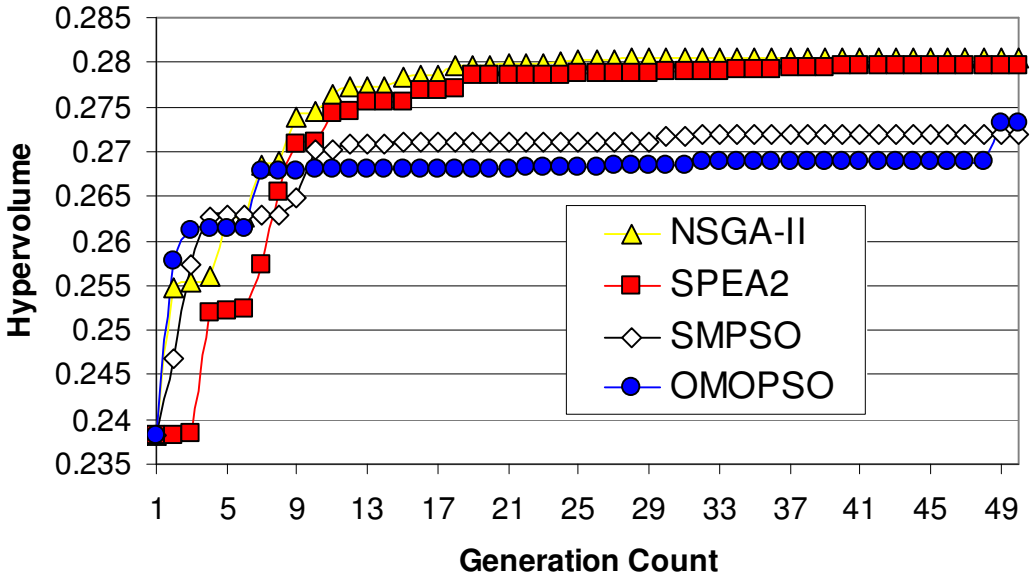

Figure 7.4-9 Hypervolume obtained by all the algorithms for the H.264 benchmark

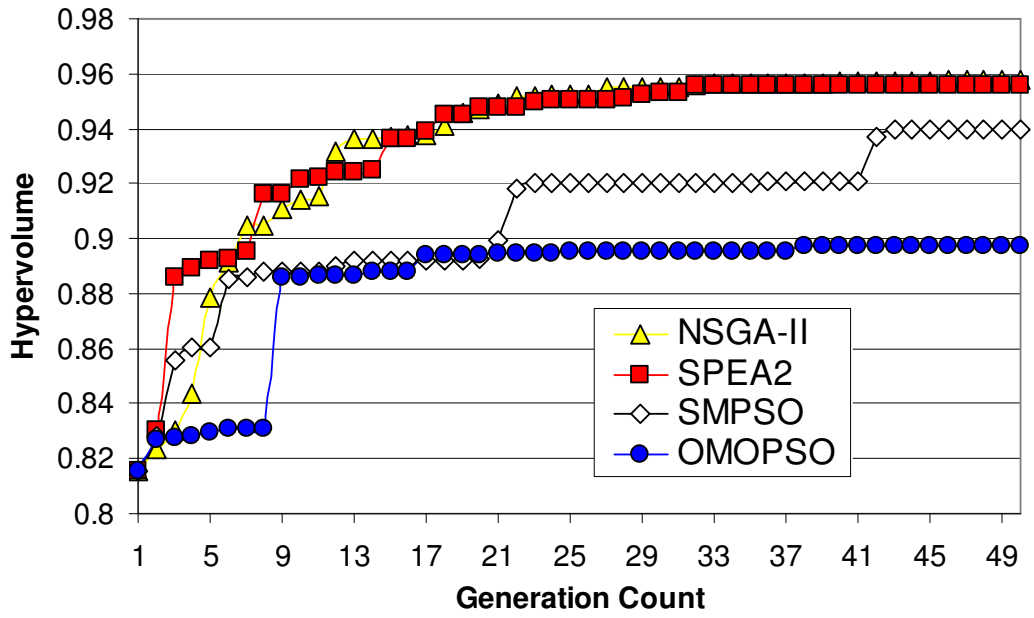

Figure 7.4-10 Hypervolume obtained by all the benchmarks for the VOPD benchmark

In both experiments the genetic algorithms perform better than the PSO ones. For the H.264 decoder benchmark the PSO algorithms have the fastest convergence speed but NSGA-II is also very close and finally the genetic algorithm even obtains better results. Of course running multiple times would have given a more correct image, but the DSE process, even with the major simplifications we proposed is still

lengthy. On VOPD, SPEA2 has the fastest convergence speed and a very good quality of results.

As a final conclusion, from all our experiments in this chapter we can say that the genetic algorithms are the best for this specific problem.

7.5 Improving the MANJAC Many-core System

The MANJAC [59] is a system with 64 native JAVA execution multi-core, multi-threaded processors arranged into an 8x8 mesh. Each processor contains 6 Jamuth [60] cores, and each core is able to run 4 threads (SMT).

We started porting a middleware on the MANJAC system called OCμ Middleware. For this, a lighter implementation that could replace the communication library used (JXTA - http://en.wikipedia.org/wiki/JXTA) had to be implemented. Special care was necessary for several reasons:

- not all the methods implemented in the Java JDK are available on Jamuth;
- the scheduler always runs the thread with the highest priority. If there are n active slots ($n = 4$) for threads then n threads can run in parallel. If there are more than n threads in the application and a thread waits for a message from a thread that it is not active a deadlock will be reached. To avoid this, the threads have to let other threads execute by going themselves to sleep;
- problems might arise when calling blocking methods (e.g. wait for a message from the network). These can also block a thread and lead to deadlocks. The calls to blocking methods have to be bounded by a timer.

More details about how to solve all these problems and about the implementation can be found in our technical report [61].

Improvements to the initialization phase for the MANJAC system were also done. We proposed new methods that allowed a better initialization phase that could cope with failing nodes in the mesh. The original implementation was prone to failures. Each node was responsible of the initialization of its North neighbor. If a node failed the entire column above him would not be initialized. We proposed two simple methods that could avoid such situations. The first method we proposed, changed the behavior of the nodes by making them send initialization messages to both North and East neighbors. The second method was a bit more advanced and was able to understand the direction from where it has received the initialization and adapt itself to where it should send itself the init messages. For this we also developed a monitoring application which allowed us to observe live, through a GUI, the initialization process.

More details about all this work can be found in our technical report "Introduction to the MANJAC system" [61].

8 Conclusions and Further Work

This work has the following contributions:

- We proposed a classification method which helped us during our experiments. The evolutionary/bio-inspired distinction was visible even in the results obtained in Chapter 7 where the algorithms grouped according to their class.
- We compared two simple evolutionary algorithms (SEMO and FEMO) on synthetic functions (LOTZ, DTLZ). We concluded that FEMO provides better results.
- We analyzed the fitness assignment process from two evolutionary algorithms (NSGA-II and SPEA2). The conclusions we drawn from this analysis were in concordance with the experimental results (SPEA2 produces more duplicates).
- We selected performance and quality metrics for multi-objective algorithms that do not require the true Pareto front to be known.
- FADSE, a tool for automatic design space exploration, was developed
- FADSE includes many design space exploration algorithms (through the integration with jMetal).
- We distributed the evaluation process with FADSE to accelerate the DSE process if resources are available. As far as we know, FADSE is the only publicly available general DSE framework for computer architectures which allows distributed evaluation.
- To run distributed the DSE algorithms had to be adapted. We changed the following algorithms: AbBYS, Densea, FastPGA, IBEA, NSGA-II, OMOPSO, PESA2, SMPSO and SPEA2.
- We provide a lot of flexibility for the allocated resources to FADSE. The number of clients can be increased/decreased dynamically during the DSE process.
- FADSE was run on different HPC systems, Linux and Windows based.
- As a further acceleration technique implemented in FADSE is the database integration. This allows us to reuse previous simulated configurations. We reached up to 67% reuse from the database. This means a great reduction in the time required for a DSE process.
- We implemented reliability techniques in FADSE: if clients do not respond, networks fail, the simulation is resubmitted to another client.
- For problems like: power loss or if the server stops responding, we implemented a checkpointing mechanism. This allows us to restart the DSE process from an intermediate state. The checkpointing mechanism can also be used to start FADSE from a user defined initial population.
- We developed an easy to use interface for connectors to the computer simulators. The work of the connector developer is simplified through different helper classes which mask the database integration and other aspects. The interface hides from the connector the DSE algorithm used and all the different settings that it can have.
- With help from our M.Sc. and B.Sc. students and collaborators we have developed connectors for the following simulators: GAP, GAPtimize, M-SIM 2, M-SIM 3, UniMap, M5 and Multi2Sim.

- Starting from the M3Explorer DSE tool we have developed our own XML configuration file. This XML is easy to use and most importantly flexible enough so that any simulator can be connected to FADSE and all the required parameters describe within this input file.
- The XML interface allows the user to configure the connector it wants to use. From this file the user can configure the database connection, benchmarks he/she wants to use, the parameters, the objectives and other relations (hierarchies) and/or constraints between them.
- Several metrics were implemented in FADSE: hypervolume, coverage, error ratio, "7 Point" average distance, etc.
- We have included into FADSE several methods to express domain knowledge: constraints, hierarchical parameters, fuzzy rules.
- Constraints were proposed and used by other DSE tools too. We observed that, when constraints are used, there tend to be many infeasible individuals in the population, which leads to a lower convergence. We changed the DSE algorithms and forced them (if the user desires) to generate at least a (user specified) percentage of feasible individuals in each offspring population. This improved the results and we used this technique in most of the experiments that employed constraints.
- In our experiments we reached a point where some parameters had to be deactivated. We decided to implement an extensible mechanism to express this knowledge. We researched the different situations that can arise when parameters depend on one another. We developed an easy to use interface to describe the hierarchies between parameters.
- The hierarchy information had to be passed to the DSE algorithms thus we proposed/developed new crossover and mutation operators.
- We propose to express domain-knowledge using fuzzy rules. This allows a designer to express general knowledge about the architecture in a close to natural language format.
- To our knowledge we are the first to use fuzzy rules as a mean to express prior knowledge into a computer systems design space exploration tool.
- We integrated *jFuzzyLogic* library into FADSE, which allows us to use the standard language FCL to describe fuzzy rules and implements multiple inference systems.
- We proposed two new mutation operators, which could take into consideration the information provided by the fuzzy rules. Both of them are derived from the classical bit flip mutation but with different methods to compute the probability to use the value obtained after defuzzification. The first method uses a constant probability to apply the information while for the other method we used a Gaussian distribution. For the second method we are not allowing it to have a probability higher than 80% and not lower than 1/number of parameters. We also using information about the membership value of the value obtained after defuzzification. We use this information as a measure of the confidence in the obtained value.
- We proposed some other enhancements. We discovered that designers do not specify the fuzzy rules at the parameter level (number of sets in the level 1 cache, block size, associativity). They sometimes prefer to use a more general notion (level 1 cache size). To allow this kind of interaction we proposed the

so called virtual parameters. Users are now allowed to combine several parameters into a single one and use that one as in input in fuzzy rules.

- We are proposing a random defuzzifier which allows the user to configure the minimum membership of the values taken into consideration. We are using this defuzzification method when the shape of the membership values defined for the rules are (close to) rectangular.
- We have proposed a method for calculating GAP's hardware complexity.
- DSE was performed on GAP with great results.
- We proved that FADSE can find better configurations than the ones found after a manual exploration of the GAP architecture. We obtained configurations with the same CPI but half the complexity. We proved that human designers can be biased and might not discover some subtle relations between the parameters.
- Single objective optimizations were conducted on code optimization tools with FADSE that led to good results.
- We performed designs space exploration of computer architectures at the same time with code optimization tools. FADSE could cope with this challenge and good results were obtained.
- We compared three DSE algorithms (NSGA-II, SPEA2 and SMPSO) and saw how they perform on the GAP architecture (with and without GAPtimize). We concluded that SMPSO is the best both in terms of convergence speed and quality of solutions. NSGA-II performed a little better than SPEA2 when only GAP was explored but the situation reversed when both GAP and GAPtimize were optimized at the same time.
- From the algorithm comparisons we concluded that in fact all the algorithms find very good results and that metrics like coverage might be misleading. We proposed to use metrics based on e-dominance.
- We proved that the integration with a database can reduce the design space exploration time to a large extent. We obtained reuse factors of over 60% in some situations.
- A method was proposed to use the fuzzy rule system integrated in FADSE with rules generated automatically from previous explorations. We showed that these rules can reduce the search time and can improve the results.
- The hierarchical parameters were tested with optimizations from GAPtimize. Good preliminary results were obtained.
- We continued the work done by Árpád Gellért in his PhD thesis, where he has performed a manual exploration of an Alpha architecture using the M-SIM 2 simulator. He varied only 2 parameters, since the design space exploration was manual. We extended this work to 19 parameters with FADSE.
- We performed different experiments with M-SIM 2 and FADSE: using constraints, starting from initial good configurations, running with information from fuzzy rules (constant and Gaussian distribution of probability to use the information).
- We showed that adding domain-knowledge can improve the results. But caution is necessary. We observed in our experiments on both GAP and M-SIM that forcing the algorithm to apply the information provided from the outside leads to a loss in diversity if the information is not diverse itself. In other words when we simulated with M-SIM and only a few rules/linguistic terms were used, applying this information with a great probability lead to a

loss in diversity and eventually not so good results. On GAP where many rules with many linguistic terms were used, the higher probability leads in fact to better results. On another experiment we introduced some good configurations in the initial population. The algorithm converged fast to very good results but it did not spread along the whole Pareto surface like the other runs did. The problem was that the good initial configurations were better than all the other individuals inserted randomly in the population, but at the same time they were very similar (differed in only 2 parameters). Being the best ones they survived for the next generations and became parents, leading to similar children. The mutation was unable to change too much the individuals and the algorithm failed to explore certain areas of the Pareto front.

- We integrated FADSE with M-SIM 3 and showed that FADSE can be used with multi-core architecture simulators.
- An overview of multi-core simulators has been performed. We also improved some of them by implementing new coherency protocols, partially integrated power consumption models, etc.
- FADSE was integrated with UniMap, a SoC simulator developed by Ciprian Radu in his PhD thesis. Together we performed automatic DSE on SoC systems. We also compared four algorithms on this new problem.
- For the SoC exploration we concluded that the genetic algorithms perform better than the PSO ones, but still, the PSO algorithms converge a little bit faster.

As future work we plan to introduce a neural network along side the DSE algorithms. As the DSE process evaluates individuals they will be used to train the neural network. When the confidence in the neural network will be high enough, it will be given random values (random values for the parameters) and it will try to predict the values of the objectives. The most promising individuals will be injected in the offspring population.

We plan to continue the work started with the fuzzy rules and propose new methods to include the information provided by them.

9 References

[1] L. Vintan, *Arhitecturi de procesoare cu paralelism la nivelul instructiunilor*. Bucharest, Romania: Editura Academiei Române, 2000.

[2] L. Vintan, "Direcţii de cercetare în domeniul sistemelor multicore / Main Challenges in Multicore Architecture Research," *Revista Romana de Informatica si Automatica*, vol. 19, no. 3, 2009.

[3] L. Vintan, *Prediction Techniques in Advanced Computing Architectures*. Bucharest, Romania: Matrix Rom Publishing House, 2007.

[4] A. Florea and L. Vintan, *Simularea şi optimizarea arhitecturilor de calcul în aplicaţii practice*. Bucharest, Romania: Matrix Rom Publishing House.

[5] H. Munk et al., "The HiPEAC Vision," *HiPEAC Roadmap*, 2010. [Online]. Available: http://www.hipeac.net/system/files/LR_3910_hipeac_roadmap-2010-v3.pdf.

[6] 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.

[7] 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*, 2002, pp. 44-53.

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

[9] **H. Calborean** and L. Vintan, "An Automatic Design Space Exploration Framework for Multicore Architecture Optimizations," in *Proceedings of The 9-th IEEE RoEduNet International Conference*, Sibiu, Romania, 2010, pp. 202-207.

[10] 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.

[11] E. Zitzler, M. Laumanns, L. Thiele, and others, "SPEA2: Improving the strength Pareto evolutionary algorithm," in *Eurogen*, 2001, pp. 95–100.

[12] M. R. Sierra and C. A. . Coello, "Improving PSO-based multi-objective optimization using crowding, mutation and e-dominance," in *Evolutionary Multi-Criterion Optimization*, 2005, pp. 505–519.

[13] A. Nebro, J. Durillo, J. Garcıa-Nieto, C. A. . Coello, F. Luna, and E. Alba, "Smpso: A new pso-based metaheuristic for multi-objective optimization," in *Proceedings of the IEEE Symposium Series on Computational Intelligence*, 2009, pp. 66–73.

[14] 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.

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

[16] E. Zitzler, "Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications," Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1999.

[17] 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.

[18] C. Silvano et al., “MULTICUBE: Multi-Objective Design Space Exploration of Multi-Core Architectures,” in *Proceedings of the 2010 IEEE Annual Symposium on VLSI*, 2010, pp. 488–493.

[19] 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, ITI-2006-10, Dec. 2006.

[20] R. Ubal, J. Sahuquillo, S. Petit, and P. López, “Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors,” in *Proc. of the 19th Int’l Symposium on Computer Architecture and High Performance Computing*, 2007.

[21] 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.

[22] E. H. Mamdani, “Application of fuzzy logic to approximate reasoning using linguistic synthesis,” *IEEE Transactions on Computers*, pp. 1182–1191, 1977.

[23] E. H. Mamdani and S. Assilian, “An experiment in linguistic synthesis with a fuzzy logic controller,” *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[24] D. T. Pham and M. Castellani, “Action aggregation and defuzzification in Mamdani-type fuzzy systems,” *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 216, no. 7, p. 747, 2002.

[25] “IEC 61131-7 standard.” [Online]. Available: http://www.fuzzytech.com/binaries/ieccd1.pdf.

[26] “International Electrotechnical Commission.” [Online]. Available: http://www.iec.ch/.

[27] P. Cingolani, “jFuzzyLogic.” [Online]. Available: http://jfuzzylogic.sourceforge.net/html/index.html.

[28] R. Jahr, T. Ungerer, **H. Calborean**, and L. Vintan, “Automatic Multi-Objective Optimization of Parameters for Hardware and Code Optimizations,” in *Proceedings of the 2011 International Conference on High Performance Computing & Simulation (HPCS 2011)*, 2011, pp. 308 – 316.

[29] **H. Calborean**, R. Jahr, T. Ungerer, and L. Vintan, “Optimizing a Superscalar System using Multi-objective Design Space Exploration,” in *Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania*, Calea Grivitei, nr. 132, 78122, Sector 1, Bucuresti, 2011, vol. 1, pp. 339–346.

[30] R. Jahr, T. Ungerer, **H. Calborean**, and L. Vintan, “Boosting Design Space Explorations with Existing or Automatically Learned Knowledge,” presented at the 16th International GI/ITG Conference on “Measurement, Modelling and Evaluation of Computing Systems” and “Dependability and Fault-Tolerance” (MMB & DFT 2012) (Submitted), Kaiserslautern, Germany, 2012.

[31] R. Jahr, “FADSE and GAP Design Space Exploration for the Grid Alu Processor (GAP) with the Framework for Automatic Design Space Exploration (FADSE),” Chamonix, France, Apr-2011.

[32] B. Shehan, R. Jahr, S. Uhrig, and T. Ungerer, "Reconfigurable Grid Alu Processor: Optimization and Design Space Exploration," in *Proceedings of the 13th Euromicro Conference on Digital System Design (DSD) 2010, Lille, France*, Los Alamitos, CA, USA, 2010, pp. 71–79.

[33] S. Uhrig, B. Shehan, R. Jahr, and T. Ungerer, "The Two-dimensional Superscalar GAP Processor Architecture," *International Journal on Advances in Systems and Measurements*, vol. 3, no. 1 and 2, pp. 71 – 81, Sep. 2010.

[34] R. Jahr, B. Shehan, S. Uhrig, and T. Ungerer, "Optimized Replacement in the Configuration Layers of the Grid Alu Processor," in *Proceedings of the Second International Workshop on New Frontiers in High-performance and Hardware-aware Computing (HipHaC'11)*, Strasse am Forum 2, 76131 Karlsruhe, Germany, 2011, pp. 9–16.

[35] R. Jahr, B. Shehan, S. Uhrig, and T. Ungerer, "Static Speculation as Post-Link Optimization for the Grid Alu Processor," in *Proceedings of the 4th Workshop on Highly Parallel Processing on a Chip (HPPC 2010)*, 2010.

[36] B. Shehan, "Dynamic Coarse Grained Reconfigurable Architectures," University of Augsburg, 2010.

[37] A. Gellert, "Advanced Prediction Methods Integrated Into Speculative Computer Architectures," "Lucian Blaga" University of Sibiu, Romania, Sibiu, Romania, 2008.

[38] A. Gellert, G. Palermo, V. Zaccaria, A. Florea, L. Vintan, and C. Silvano, "Energy-performance design space exploration in SMT architectures exploiting selective load value predictions," in *Proceedings of the Conference on Design, Automation and Test in Europe*, Dresden, Germany, 2010, pp. 271–274.

[39] A. Gellert, A. Florea, and L. Vintan, "Exploiting selective instruction reuse and value prediction in a superscalar architecture," *Journal of Systems Architecture*, vol. 55, no. 3, pp. 188–195, Mar. 2009.

[40] J. J. Sharkey, D. Ponomarev, and K. Ghose, "M-SIM: A Flexible, Multithreaded Architectural Simulation Environment - Technical Report CS-TR-05-DP01." Department of Computer Science, State University of New York at Binghamton, Oct-2005.

[41] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," *SIGARCH Comput. Archit. News*, vol. 25, no. 3, pp. 13–25, Jun. 1997.

[42] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," in *Proceedings of the 22nd annual international symposium on Computer architecture*, S. Margherita Ligure, Italy, 1995, pp. 392–403.

[43] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*, Vancouver, Canada, 2000, vol. 28, pp. 83–94.

[44] "The SPEC benchmark programs." [Online]. Available: http://www.spec.org.

[45] A. Gellert, **H. Calborean**, L. Vintan, and A. Florea, "Multi-Objective Optimizations for a Superscalar Architecture with Selective Value Prediction," *IET Computers & Digital Techniques (submitted, manuscript ID CDT-2011-0116).*

[46] D. August et al., "Unisim: An open simulation environment and library for complex architecture design and collaborative development," *Computer Architecture Letters*, vol. 6, no. 2, pp. 45–48, 2007.

[47] C. Bienia, “Benchmarking Modern Multiprocessors,” Princeton University, 2011.
[48] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in *Proceedings of the 22nd International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, 1995, pp. 24–36.
[49] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, “The M5 Simulator: Modeling Networked Systems,” *IEEE Micro*, vol. 26, pp. 52-60, 2006.
[50] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 469–480.
[51] J. Renau et al., *SESC simulator*. 2005.
[52] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, “HotSpot: A compact thermal modeling methodology for early-stage VLSI design,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 5, pp. 501–513, 2006.
[53] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, “A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies,” in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, 2008, pp. 51-62.
[54] C. Radu and L. Vinţan, “UNIMAP: UNIFIED FRAMEWORK FOR NETWORK-ON-CHIP APPLICATION MAPPING RESEARCH,” *Acta Universitatis Cibiniensis â€“ Technical Series*, May 2011.
[55] C. Radu, “Optimized Algorithms for Network-on-Chip Application Mapping,” PhD thesis, “Lucian Blaga” University of Sibiu, Romania, Sibiu, Romania, 2011.
[56] C. Radu, “Developing Network-on-Chip Architectures for Multicore Simulation Environments,” PhD Technical Report no. 1, Computer Science Department, “Lucian Blaga” University of Sibiu, PhD report 1, Jun. 2010.
[57] “The Embedded System Synthesis Benchmarks Suite (E3S) website.” [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s/.
[58] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, “ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, 3001 Leuven, Belgium, Belgium, 2009, pp. 423–428.
[59] S. Uhrig, “The MANy JAva Core processor (MANJAC),” in *HPCS*, 2010, p. 188.
[60] S. Uhrig and J. Wiese, “jamuth: an IP processor core for embedded Java real-time systems,” in *Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems*, New York, NY, USA, 2007, pp. 230–237.
[61] **H. Calborean**, “Introduction to the MANJAC system,” Computer Science Department, “Lucian Blaga” University of Sibiu, Sibiu, Romania, 4, 2011.