"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

Universitatea “Lucian Blaga” din Sibiu
Facultatea de Inginerie “Hermann Oberth”
Catedra de Calculatoare şi Automatizări

Optimizarea multi-obiectiv a unor arhitecturi avansate de calcul utilizând cunoştinţe de domeniu

Teză de doctorat

Autor:
Ing. Horia Andrei Calborean

Conducător ştiinţific:
Prof. univ. dr. ing. Lucian Vinţan

SIBIU, Septembrie 2011

Mulţumiri

În primul rând îi mulţumesc conducătorului meu de doctorat, domnul profesor *Lucian Vinţan*, pentru încrederea acordată si pentru oportunitatea de a lucra în acest proiect de cercetare. Mulţumiri sincere pentru faptul că a fost alături de mine în această aventură şi m-a ajutat, prin sfaturile date, să o duc la bun sfârşit.

Aş vrea să-i mulţumesc profesorului *Theo Ungerer* de la Universitatea din Augsburg pentru stagiul de pregătire de 5 luni pe care mi l-a oferit în cadrul grupului de cercetare condus de domnia sa. Apreciez in mod special colaborarea dintre mine şi membrii acestei echipe, care a continuat şi după întoarcerea mea în ţară. Doresc să-i mulţumesc pe această cale lui *Ralf Jahr* împreună cu care am obţinut rezultate deosebite şi am ajuns să devenim prieteni.

Îi mulţumesc lui *Ciprian Radu*, prietenul şi colegul meu, pentru sfaturile şi sprijinul acordat în toată această perioadă.

Mulţumesc membrilor Catedrei de Calculatoare si Automatizări de la Universitatea "Lucian Blaga" din Sibiu, în special domnului conferenţiar univ. dr. ing. *Remus Brad* care mi-a recenzat o parte din munca desfăşurată în această perioadă.

Aş dori să îi mulţumesc domnului conferenţiar univ. dr. ing. *Adrian Florea* şi domnului asistent univ. dr. ing. *Árpád Gellért* pentru sprijinul acordat, comentariile constructive şi pentru buna colaborare avută în ultimii ani, chiar dinainte de începerea perioadei doctorale.

Sincere mulţumiri echipei conduse de domnul profesor *Nicolae Ţăpuş* pentru că mi-a permis accesul la sistemul HPC din cadrul Universităţii Politehnice din Bucureşti. Îi menţionez pe această cale pe domnul conferenţiar univ. dr. ing. *Emil Slusanschi* şi pe domnul asistent univ. *Alexandru Herişanu* cărora le mulţumesc pentru ajutorul oferit.

În această cercetare am colaborat cu studenţi din cadrul Universităţii "Lucian Blaga": *Andrei Zorilă, Camil Băncioiu* şi *Radu Chiş*. Doresc să le mulţumesc pentru ajutorul dat.

În cele din urmă vreau să mulţumesc câtorva persoane foarte dragi mie: soţiei mele *Angela* şi părinţilor noştri pentru că au fost alături de mine şi m-au sprijinit în toată această perioadă.

Cercetările prezentate în lucrare au fost realizate în cadrul proiectului POSDRU 7706: *Creşterea rolului studiilor doctorale şi a competitivităţii doctoranzilor într-o Europă unită* cofinanţat din Fondul Social European prin Programul Operaţional Sectorial Dezvoltarea Resurselor Umane 2007 - 2013.

Rezumat

Pe măsură ce tehnologia a avansat, sistemele informatice au devenit tot mai complexe [1] [2] [3] [4]. Când se proiectează un astfel de sistem, arhitectul trebuie să ţină cont de o mulţime de parametri astfel încât spaţiul de proiectare a asamblului microprocesor-compilator poate ajunge la milioane de miliarde de configuraţii posibile. Un microprocesor nu poate intra în producţie până când nu este evaluat pentru a îndeplini criteriile de performanţă şi corectitudine. Evaluarea unei singure configuraţii poate dura ore sau chiar zile, prin urmare, o evaluare exhaustivă devine imposibilă.

Abordarea actuală este de a utiliza experţi umani care selectează configuraţii, considerate bune, pe care apoi le evaluează folosind simulatoare specializate. Odată cu creşterea în complexitate şi cu creşterea numărului de nuclee eterogene integrate, sarcina designer-ului de a găsi configuraţii bune devine tot mai grea. Problema este exacerbată de faptul că mai multe obiective trebuie optimizate simultan: performanţa, consumul de putere, aria de integrare etc. Descoperirea relaţiilor între parametrii unei arhitecturi de calcul şi modul în care influenţează ele obiectivele se dovedeşte a fi o sarcină dificilă, aproape imposibil de realizat manual.

O soluţie la această problemă este utilizarea de instrumente care efectuează automat o explorare a spaţiului de proiectare, folosind diferiţi algoritmi euristici de căutare. În viziunea HIPEAC [5], explorarea automată a spaţiului configuraţiilor (design space exploration - DSE) este una dintre cele mai importante probleme care trebuie rezolvate în următorii ani. Algoritmii de căutare euristică nu sunt o noutate in domeniu, aceştia fiind folosiţi pentru rezolvarea de probleme NP-hard, dar în ultima perioadă li se acorda un interes tot mai mare, fiind una din puţinele soluţii viabile pentru DSE.

Scopul acestei teze este de a explora automat spaţiul de proiectare a unor arhitecturi de calcul şi de a îmbunătăţi algoritmii utilizaţi prin cunoştinţe avansate de domeniu. Pentru aceasta trebuie îndeplinite următoarele **obiective**:

- Analizarea algoritmilor euristici multi-obiectiv folosiţi pentru DSE;
- Analizarea comportamentului acestora în probleme reale (în cazul în care există constrângeri între parametrii);
- Găsirea unor metrici de evaluare a performanţelor acestor algoritmi;
- Dezvoltarea unei aplicaţii software robuste şi rapide, care să se poată conecta la orice simulator de arhitecturi de calcul existent;
- Integrarea mai multor algoritmi euristici în această aplicaţie astfel încât să se poată realiza cu uşurinţă o comparare a acestora;
- Cercetarea modului în care cunoştinţele de domeniu pot fi integrate în aplicaţia noastră şi cum pot ele pot influenţa performanţa algoritmilor euristici utilizaţi;
- Testarea şi evaluarea algoritmilor pe diferite tipuri de arhitecturi (single-core, multi-core, System on Chip etc.);
- Compararea algoritmilor şi determinarea impactului integrării cunoştinţelor de domeniu asupra rezultatelor obţinute.

Capitolul 2 prezintă câţiva algoritmi de căutare bine cunoscuţi, frecvent utilizaţi în probleme de optimizare multi-obiectiv, pe care i-am integrat în aplicaţia noastră;

Capitolul 3 descrie principalele caracteristici ale uneltei software dezvoltată pentru explorarea automata a spaţiului de proiectare. Acest instrument este denumit FADSE (*Framework for Automatic Design Space Exploration*). Scopul lui este de a accelera procesul de DSE, nu doar prin utilizarea algoritmilor euristici, ci şi prin permiterea evaluării paralele a configuraţiilor. Aplicaţia integrează şi o bază de date care îi permite reutilizarea rezultatelor obţinute anterior (indivizi deja simulaţi), conducând deci la o scăderea a timpului necesar pentru explorare.

În Capitolul 4 introducem o metodă nouă de accelerare a procesului de explorare şi de creştere a calităţii rezultatelor. Toţi algoritmii de căutare folosiţi în această lucrare sunt algoritmi generali şi pot fi folosiţi pentru aproape orice problemă de căutare. I-am modificat cu scopul de a utiliza informaţii date sub formă de reguli fuzzy de către un expert uman. Utilizatorul trebuie să descrie parametrii folosindu-se de termeni lingvistici (de exemplu: un cache de nivel unu mai mare de 256KB este „mare", pentru valori sub 64KB este „mic" etc.) şi apoi să introducă reguli uşor de înţeles, cum ar fi: dacă nivelul 1 de cache este mic, atunci nivelul 2 de cache trebuie să fie mare. Informaţiile furnizate prin aceste reguli sunt luate în considerare în timpul procesului de explorare, pentru a ghida căutarea. Pe lângă această metodă, mai propunem si alte tehnici prin care cunoştinţele expertului pot fi integrate într-un mod facil în algoritmii de căutare: constrângeri între parametri, ierarhii de parametri.

Capitolul 5 prezintă rezultatele testelor efectuate cu FADSE pe simulatorul procesorului GAP [6] împreună cu un optimizator de cod dezvoltat special pentru acest procesor, denumit GAPtimize. După analizarea rezultatelor am ajuns la concluzia că FADSE a descoperit configuraţii mai bune decât cele găsite de către un expert uman prin explorare manuală. De asemenea, am arătat că FADSE este scalabil şi capabil de a găsi rezultate foarte bune în spaţiile de proiectare extrem de mari generate de parametrii simulatoarelor. Am realizat o comparaţie între diferite tipuri de algoritmi de căutare pentru a determina care algoritm oferă rezultatele cele mai bune. Mai prezentăm şi o metodă de a obţine în mod automat reguli fuzzy pe baza explorărilor anterioare.

Capitolul 6 se concentrează asupra familiei de simulatoare M-SIM. Am extins munca domnului Dr. ing. Árpád Gellért [6]. În teza sa de doctorat (realizată tot sub conducerea ştiinţifică a d-lui Profesor Lucian Vinţan) s-au variat doar 2 din cei 19 parametri (schimbarea tuturor acestor parametri manual ar fi fost o sarcină foarte dificilă şi consumatoare de timp). Am folosit FADSE pentru a varia toţi parametrii şi am găsit configuraţii mult mai bune pentru procesorul (*Alpha*) simulat. Am încercat diferite metode pentru a accelera procesul de DSE. Una dintre ele este introducerea în populaţia iniţială a celor mai bune configuraţii găsite manual, pornind astfel de la o poziţie mai buna in spaţiul de căutare. O altă metodă este utilizare de reguli fuzzy dezvoltate pe baza observaţiilor din experimentele anterioare.

În Capitolul 7 ne-am axat pe un domeniu total diferit: SoC (System on Chip). Am folosit un simulator, dezvoltat de către ing. Ciprian Radu pentru teza sa de doctorat (conducător ştiinţific Prof. Lucian Vinţan), denumit UniMap. Am testat mai mulţi algoritmi DSE pentru a afla care obţine rezultatele cele mai bune. Am obţinut rezultate diferite fata de cele din comparaţiile efectuate în Capitolul 5. Acest lucru ne-a condus la concluzia că alegerea celui mai bun algoritm depinde de problema care trebuie rezolvată.

Teoria prezentată în această lucrare (clasificări, definiţii, glosar) este strict subordonată scopului si obiectivelor practice ale acestei teze. Cu alte cuvinte, nu am încercat o prezentare exhaustivă şi nici nu am pretenţia unei rigori general valabile.

"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.

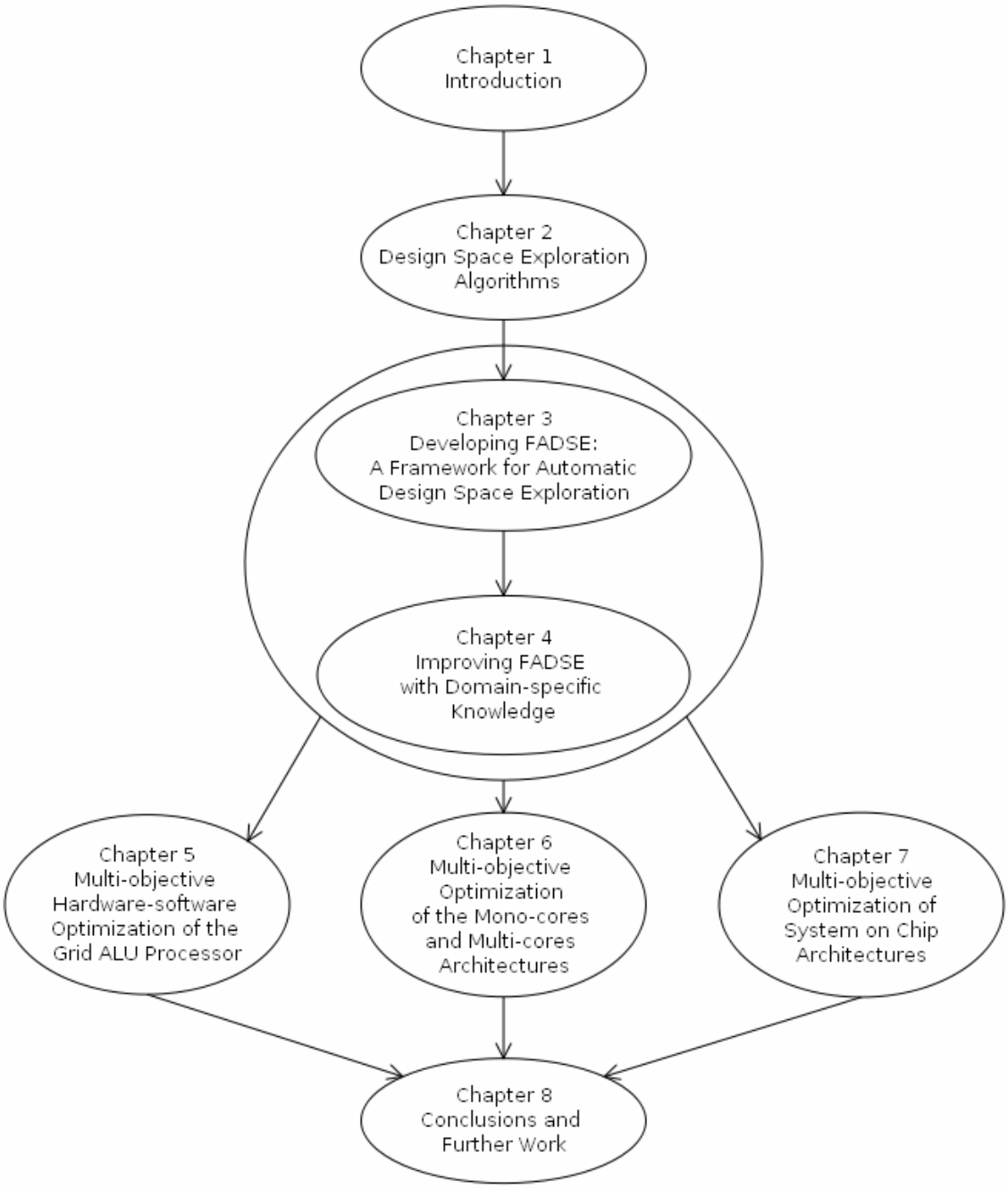
For this research I have collaborated with very good students from "Lucian Blaga" University of Sibiu: *Andrei Zorila, Camil Banciou* and *Radu Chis*. I would like to thank them for their help.

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

Thesis Structure



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.

In Chapter 2 we present some well-known search algorithms that are frequently used in multi-objective optimization problems which we integrated in our tool presented in Chapter 3.

Chapter 3 describes the main features of our framework that we implemented for ADSE. It is called FADSE (from *Framework for Automatic Design Space Exploration*). It includes the algorithms presented in Chapter 2 and many more. The main aim of this tool is to accelerate the DSE process, not only by using heuristic algorithms but also through parallel evaluation. The integration of a database allows the reuse of already simulated individuals and decreases the time required for an exploration.

In Chapter 4 we introduce a novel method to accelerate the DSE process and increase the quality of the results. All the search algorithms used in this work are general algorithms and they do not have any knowledge about the problem they solve. We changed the algorithms to accept information in form of fuzzy rules from a human expert. The user has to describe the parameters using linguistic terms (e.g. a level 1 cache higher than 256KB is big, for other values it is small, etc.) and then write some easily understandable rules such as: if level 1 cache is small then level 2 cache is big. The information provided by these rules is used during the DSE process to guide the search. We also integrate an easy to use interface through which the designer can specify constraints of the problem and/or relations between the parameters (hierarchies).

Chapter 5 presents the exploration performed with FADSE on the simulator of the GAP processor [6]. The work is then extended to the code optimization tool developed for GAP, called GAPtimize. After analyzing the results, we concluded that the configurations found by FADSE are better than the ones found by a human expert through manual exploration. We also show that FADSE is scalable and it finds very good results in extremely large design spaces. A comparison between different search algorithms is performed. We are also presenting a method of automatically obtaining fuzzy rules from previous explorations. These rules can be used to accelerate future design space explorations.

Chapter 6 focuses on the M-SIM family of simulators. We are continuing the work performed by Dr. Árpád Gellért in his PhD thesis developed under Professor Vintan's supervision [6]. In this previous work a manual exploration was performed on the M-SIM 2 simulator. Since Dr. Árpád Gellért was able to vary only 2 of the 19 parameters, it was obvious that better configurations can be found. Changing all these parameters manually would have been a very difficult and time consuming task. We employed FADSE for this assignment. We tried different methods to accelerate the design space exploration. One of them is to insert the manually found configurations in the initial population, thus giving the algorithm a head start. Another one was to benefit from our previous experience and designed some fuzzy rules which were used to accelerate the design space exploration. We analyzed the results obtained and concluded that adding information can help the search algorithm in finding faster better results.

In Chapter 7 we switch to a slightly different domain: SoC (System On Chip). We are using a simulator, developed by Ciprian Radu for his PhD thesis, called UniMap. We test multiple DSE algorithms on this problem to determine the best one. Interesting results are obtained, different from to the results obtained from the comparison performed in Chapter 5. This led us to the conclusion that choosing the best algorithm is dependent on the problem that needs to be solved.

The theory presented in this work (classifications, definitions, and glossary) is strictly governed by the scope and the practical objectives of this work. In other words it is not intended to provide an exhaustive overview of the domain.

"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 search problems. Manual design space exploration is not feasible so new methods are required. One of them is to use algorithms to perform this task automatically. In this chapter we are presenting some possible approaches and then focus on the ones we used throughout this thesis.

2.1 Classification

2.1.1 Exhaustive Search or Heuristic Algorithms

We divide the search algorithms into two main categories: exhaustive algorithms (or enumerative algorithms) and heuristic algorithms.

An exhaustive search algorithm simply tries all the possible combinations. Due to the enormous search space they are practically prohibited. We will not discuss them any further.

Heuristic algorithms use a previous assumption of the designer. A simple example is greedy search where the designer of the algorithm has decided that the road which seems the best at a point in time is selected. This approach does not guarantee that the optimal solution is found, but from experience and sometimes with mathematical proof, we seen that they provide good solutions.

2.1.2 Deterministic or Stochastic

Further, we divide the heuristic methods into two classes: deterministic and stochastic.

Deterministic algorithms always behave the same. They are considered heuristic methods because they attempt to incorporate some sort of knowledge. Most of these algorithms are graph/tree search algorithms. Some of these methods are: greedy algorithm, hill climbing, branch and bound methods, depth-first and breadth-first methods, best-first methods (A^*, Z^*).

The next classes of methods are the stochastic ones. These methods incorporate a random behavior in them. They were developed to avoid problems encountered by the deterministic methods, such as local minima. A simple example is between hill-climbing (deterministic) and simulated annealing (stochastic). Hill-climbing always chooses the best solution. When it is not able to find any better solution it stops. Simulated annealing can select randomly a worse solution at specific times, allowing it to escape from local minima.

This work will use only on stochastic-heuristic methods. There are several algorithms that fall in this category: simulated annealing, evolutionary algorithms, etc. We will focus on the latter.

2.1.3 Evolutionary Algorithms

This classification is governed by the scope of this work and not a general one. We split the heuristic algorithms into two classes: genetic algorithms and bio-inspired algorithms.

2.1.3.1 Genetic Algorithms

Genetic algorithms are well known for their capacity to find good solutions for NP-hard problems. This work we will not provide an overview of the existing implementations. We will focus only on the algorithms we used in our experiments. Many resources that detail how the genetic algorithms work can be found. In Figure 2.1-1 we present the basic structure of a canonical genetic algorithm.

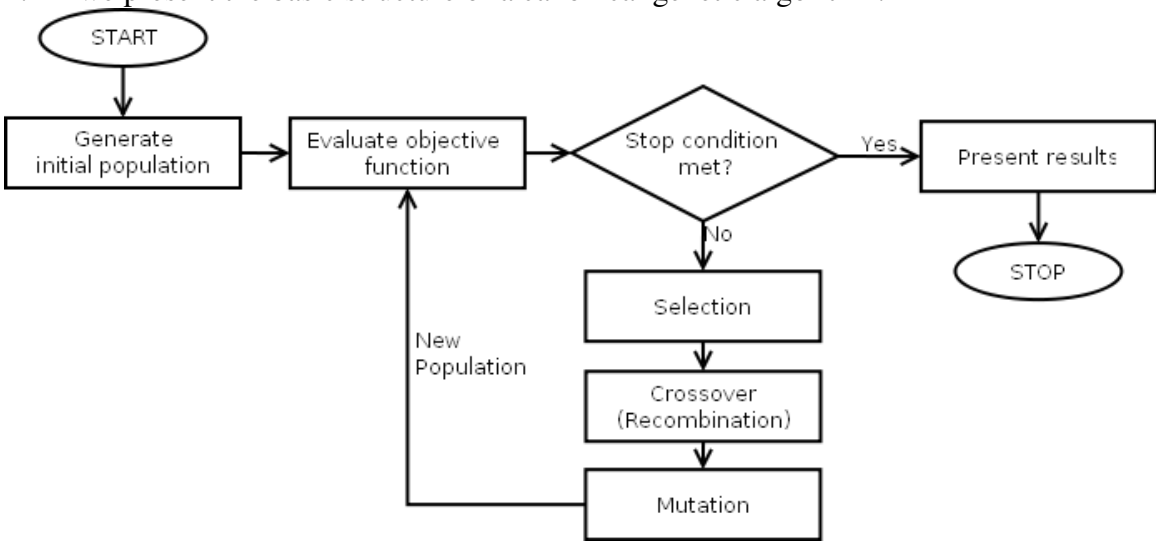


Figure 2.1-1 Structure of a population based evolutionary algorithm

The algorithm usually starts from a random initial population of individuals and it runs until a stop condition is reached: a good enough solution has been found, time expired, etc. After the initial population is evaluated the genetic algorithm enters the main loop. If the stop condition has not been met, it enters the selection phase where some individuals are selected from the entire population to become parents. They will produce offspring through crossover and/or mutation. The produced individuals will form the offspring population which will get evaluated. Depending on the algorithm a new parent population is obtained: simply by using the offspring as parents or, more commonly, trough other mechanisms that combine the offspring and the former parent population (like elitism).

Further we will present the commonly used operators for selection, crossover and mutation. We will focus only on the ones used in our experiments.

Selection Operators

First we describe tournament selection operator, from which the operator we used is developed.

In tournament selection, a few individuals are selected randomly from the population. The best one from these selected individuals becomes a parent.

For selection operators it is important to talk about selection pressure. The selection pressure is the degree the better individuals are favored by the operator. The higher the selection pressure, the higher the chance that the selected individual is better.

In tournament selection, the selection pressure can be modified by changing the number of individuals selected for tournament. If the number is high, a weak individual has a lower chance to become a parent.

The most used selection method in this work is the binary tournament selection. In this method only two individuals are selected for tournament (low selection pressure). The better of these two will become a parent.

Crossover Operators

Usually two individuals are used to perform crossover. The simplest implementation is the single point crossover operator. The individuals are seen as an array of elements. A random point is chosen that cuts both individuals in the same place. This point separates the individual in two parts. The offspring will be formed from one part of the first parent and the second part from the other parent.
This crossover operator can be extended to work with multiple crossover points.

Mutation Operators

Only one individual is needed for mutation. We focus only on the operators used in this work. The most used one is the bit flip mutation operator (for integers). For this operator the user has to specify a probability of mutation. This will determine how many of the parameters from an individual will be changed. The pseudo code for the bit flip mutation is shown below:

1. For all the parameters in the individual;
   1.1. Generate a random number between 0 and 1;
   1.2. If the random number is smaller than the probability of mutation;
      1.2.1. Change the current parameter to a random value;
2. STOP.

Other operators we used are: uniform mutation, non-uniform mutation and polynomial mutation [7]. The uniform mutation is similar with the bit flip mutation for integers but besides the mutation probability a perturbation parameter has to be specified. Depending on this perturbation the new value for the parameter is closer of farther from the current value.

The non-uniform mutation is similar with the uniform mutation. The difference is that at the beginning of the algorithm the values of the parameters to be mutated can be at a greater distance from the initial value (greater perturbation). Polynomial mutation is similar but the value of the selected parameter to be mutated is changed according to a polynomial probability distribution.

2.1.3.2 Bio-inspired Algorithms

From the bio-inspired algorithms, we have used some Particle Swarm Optimization (PSO) methods. PSO is a population based stochastic optimization technique first proposed in [8] and it was inspired by the bird flocking.

In PSO the individuals (called particles in the algorithm) “fly” in the design space following the current best individual.

The swarm is typically modeled by individuals in a multidimensional space that have a position and a velocity associated. The individuals evaluate themselves and remember the location where they had their best success. This value is known as the *personal best*. All the individuals are informed about the best individual found so far which is called the *global best*. The movement through the search space is guided by these previous successes. Each of the individuals adjusts its position and velocity based on this information.

After finding the local and global best values, the individual *i* updates its velocity $\vec{v}_i(t)$ and position $\vec{x}_i(t)$ with following equations:

Equation 2-1

$$\vec{v}_i(t) = W\vec{v}_i(t-1) + C_1 r_1(\vec{x}_{pbest_i} - \vec{x}_i(t-1)) + C_2 r_2(\vec{x}_{gbest} - \vec{x}_i(t-1))$$

Equation 2-2

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t)$$

where $\vec{v}_i(t)$ is the particle velocity, $\vec{x}_i(t)$ is the current particle (individual). $\vec{x}_{pbest_i}$ and $\vec{x}_{gbest}$ are defined as stated before. r_1 and r_2 are random numbers between (0,1). C_1, C_2 are learning factors. C_1 is the cognitive learning factor and C_2 is the social learning factor. Usually $C_1 = C_2 = 2$. *W* is called the inertia weight. It is used to control the impact of the previous history of velocities on the current velocity. In the standard algorithm *W=1*.

The algorithm is presented below:

1. Initialize the individuals;
2. If the maximum number of iterations has been reached or another criteria is met jump to step 3;
   2.1. Evaluate all the individuals;
      2.1.1. If the new value is better than its personal best, remember the current value as the personal best;
   2.2. Find the individual with the best value and set it as the global best;
   2.3. For each individual compute its new velocity using Equation 2-1 and its position using Equation 2-2;
   2.4. Jump to step 2;
3. STOP.

Usually the velocity is limited to a maximum value V_{max} specified by the user. If the velocity exceeds this value then it is limited to V_{max}.

The algorithm is similar with the genetic algorithms: generates an initial population formed from random individuals and then searches for a solution by updating this population in each generation. The difference from genetic algorithms is that in the PSO algorithm there is no selection mechanism. The absence of a selection mechanism in PSO is compensated by the use of leaders to guide the search. There is no notion of offspring generation in PSO as with evolutionary algorithms.

In later versions of the algorithm, the notion of neighborhood was introduced. A *local best* value was stored for the best particle in the neighborhood of the current particle. Different topologies of the neighborhood were introduced to determine the neighbors of any given particle. This is necessary for finding the local best. Possible topologies are presented in [9].

The PSO algorithms usually have a fast convergence. This can lead to premature convergence (converging to local minima). To avoid these situations, methods for maintaining diversity are needed. Convergence is influenced by the topology, so this is one way to slow down the convergence speed. Another way is through the position updating mechanism. When the population stagnates turbulence might be infused in the population [10] which might help the algorithm escape the

local minima. But this raises other problems some of them are further presented in [9]: choosing a good turbulence operator, choosing the turbulence probability, choosing at what moment to apply the turbulence and on which particle

The inertial weight *W* in Equation 2-1 could be adjusted like in a simulated annealing algorithm. A large inertial weight facilitates the global exploration. The new velocity will not have such a great impact and the particles will not tend to follow global best. A small inertial weight will facilitate local exploration.

2.1.4 Single and Multi-objective Algorithms

Another classification we make depends on the number of objectives a problem can have. There are single-objective problems and multi-objectives. All the algorithms described in previous chapters are single objective ones and they can solve problems where the evaluation produces a single output. This thesis is focused on the multi-objective ones.

2.2 Multi-objective Optimization

Multi-objective optimization is the process of simultaneously optimizing two or more (usually) conflicting objectives. Not only in computer architecture, but in many of the real world problems, the optimizations, that have to be carried out frequently, have more than just one objective. Usually a trade-off must be made between two or more conflicting objectives.

In most multi-objective optimizations there are many solutions to the problem. There is no single solution that simultaneously minimizes/maximizes each objective to the fullest. Multi-objective optimization algorithms are looking for solutions for which each objective has been optimized to a maximum extent. If we try to optimize it any further, then the other objective(s) will suffer as a result. Finding such solutions is the goal when setting up and solving a multi-objective optimization problem.

Some of the search methods, used in multi-objective optimizations, are: simulated annealing, genetic algorithms, particle swarm optimization. The multi-objective algorithms are derived from the single objective ones. Usually they attack to major issues: assigning an order (or a fitness value) to the individuals before selection and preserving diversity. These two make the multi-objective algorithms usually different from the single objective variants.

2.2.1 Classification

2.2.1.1 Aggregating Approaches

Algorithms that fall in this category combine (or aggregate) all the objectives of the problem into one single objective. The effect is that the multi-objective problem is transformed into a single objective one and single objective algorithms can be used.

Under this category we can find: weighted-sum method, method of goal attainment. The disadvantage of these methods is that a single solution is provided as output. Also, sometimes a fair aggregation is impossible.

2.2.1.2 Lexicographic Ordering

In this method the user (decision maker) has to rank all the objectives in order of their importance. Then the algorithm optimizes the first objective, then with the found solution tries to optimize the second objective, and so on. It is acting like hill-climbing algorithm. This method is sensitive to the ordering of the objectives.

2.2.1.3 Sub-population Approaches

In this approach several instances of a single objective algorithm run in parallel and try to optimize one of the objectives. At certain periods of time, the algorithms exchange information. If the algorithm is a population based algorithm, the population is divided in several subpopulations. These subpopulations are used with single objective optimizers. Then, the subpopulations exchange information or recombine among themselves aiming to produce trade-offs among the different solutions previously generated for the objectives that were separately optimized.

2.2.1.4 Pareto-based Approaches

These approaches use individual selection techniques based on Pareto dominance. The basic idea of all the approaches considered here is to select the individuals that are nondominated by other individuals.

2.2.2 Pareto Efficiency

Pareto efficiency, or Pareto optimality, is an important concept in economics with applications in game theory, engineering and social sciences.

Pareto efficient: situations are those in which any (additional) change to make any individual better off is impossible without making someone else worse off.

Pareto improvement: if a change to make an individual better does not make any other individual worse.

Pareto efficient or **Pareto optimal:** when no further Pareto improvements can be made. This means that any additional change will make another individual worse. This is also called **strong Pareto optimum (SPO)** to separate it from the "weak Pareto optimum" defined below.

Weak Pareto optimum (WPO): the difference from SPO is that a weak Pareto improvement is considered when **all** the individuals gain from that change. Any SPO is a WPO, because any change in the SPO will not improve any individual. A WPO is not a SPO, since a change in WPO might improve the individual and leave the others the same.

Pareto front or **Pareto set** is the front/set of choices that are Pareto efficient.

In Figure 2.2-1 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.

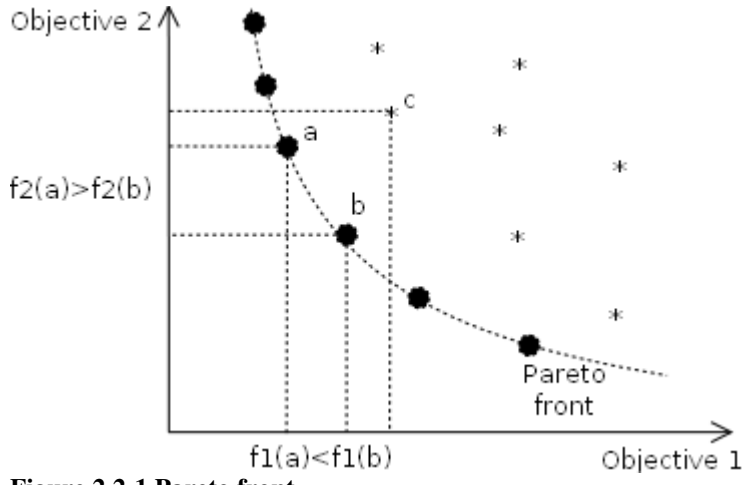


Figure 2.2-1 Pareto front

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 [9]:

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

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

2.3 Multi-objective Evolutionary Algorithms

2.3.1 Simple Multi-objective Algorithms

Simple Evolutionary Multi-objective Optimizer (SEMO)

The SEMO algorithm [11] is one of the simplest algorithms used for multi-objective optimization. It stores an archive of all the non-dominated individuals. This archive represents the current population. From this population a parent is chosen (usually random) and mutated by randomly changing a chosen bit. 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.

Fair Evolutionary Multi-objective Optimizer (FEMO)

The FEMO algorithm [11] is an improvement of SEMO. The problem is that some individuals are chosen more often than others to become parents. Because of this, some individuals will get mutated more often just because they were added to the population earlier in history. The consequence is that the neighborhood of these individuals will get over explored while others will not. The FEMO algorithm tries to avoid this situation by counting the number of offspring each individual produces. When selecting the parent, the algorithm deterministically chooses the individual with the smallest number of offspring. If more than one individual with the same number of offspring is found then the parent is chosen randomly between these individuals.

2.3.2 NSGA-II

The Nondominated Sorting Genetic Algorithm (NSGA) proposed in [12] was one of the first genetic algorithms able to find multiple Pareto-optimal solutions. In the following years the importance of elitism was demonstrated. Elitism can speed up the genetic algorithm significantly and also prevents the loss of good solutions once they are found. NSGA does not take into consideration this aspect. NSGA was criticized for being too computationally intensive for large population sizes. Its nondominated sorting algorithm has a computational complexity of $O(MN^3)$ where M is the number

of objectives and N is the population size. The fact that the designer had to specify the σ_{share} parameter was also a problem for NSGA. The σ_{share} parameter influenced the fitness assignment process and had a great impact on the performance of the algorithm. NSGA-II [13] solves all these mentioned problems.

The nondominated sorting approach

For each individual the domination count (number of individuals that dominate the current individual) and a set of individuals dominated by the current individual are computed. The individuals from the first front will have their domination count equal to zero. Each individual from this front is taken separately and the individuals that it dominates are added to a set. The domination count of the individuals from this set is decremented by one. The process is repeated until all the individuals from this front are analyzed. Now all the individuals from the set who have their domination count equal with 0 will constitute de new front (the second front). The process is repeated until all the individuals are sorted into fronts.

Diversity preservation

The NSGA algorithm used the sharing function approach to preserve diversity. This requires the specification of σ_{share}. The σ_{share} represents the maximum distance between two individuals, so that they are considered members of the same cluster. The idea is to maintain individuals in less crowded clusters. Usually, σ_{share} had to be specified by the designer. This greatly influenced the performance of the algorithm. To overcome these problems, in NSGA-II a new algorithm for preserving diversity is proposed. In NSGA-II no parameter has to be specified for maintaining diversity in the population. First two notions have to be defined: density estimation and crowded-comparison operator.

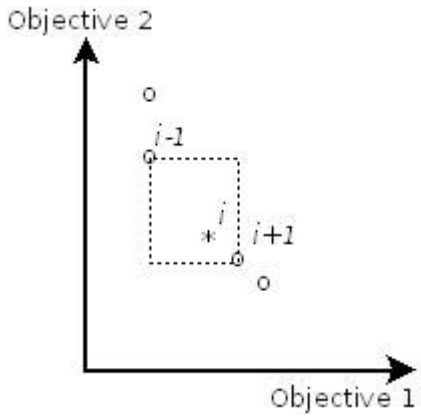


Figure 2.3-1 Density estimation

Density estimation

For each point, its nearest two neighbors are searched. These neighbors have to be one on one side of the node and the other on the other side relative to one objective (see Figure 2.3-1). The points are sorted according to the first objective. The *i* counter represents the position in the sorted vector. The *crowding distance* = $I_{distance}$ is obtained by adding the distances, computed using the formula below, for each objective:

$$I[i]_{dis\tan ce_m} = \frac{I[i+1]_{dis\tan ce_m} - I[i-1]_{dis\tan ce_m}}{f_m^{\max} - f_m^{\min}}$$

where $I[i]_{dis\tan ce_m}$ represents the m[th] objective function value of the i[th] individual in the I front. The parameters $f_m^{\max}$ and $f_m^{\min}$ are the maximum and minimum values of the m[th] objective function. The individuals at the extremes are assigned with infinite values ($I[0]_{dis\tan ce} = \infty$ and $I[\max]_{dis\tan ce} = \infty$). An individual with a smaller value of this distance is in fact a more crowded individual.

Crowded-comparison operator

The crowded-comparison operator $\prec_n$ guides the selection process. Each individual in the population has two attributes associated: its non-domination rank and its crowding distance. Non-domination rank represents the current front number. Crowding distance was defined above.

if ($i_{rank} < j_{rank}$) $i \prec_n j$

if ($i_{rank} = j_{rank}$) $i \prec_n j$ if ($i_{distance} > j_{distance}$)

This means the algorithm prefers an individual from a lower (better) front, a front that dominates more individuals. If the individuals are on the same front, then a less crowded individual is preferred.

NSGA-II algorithm

1. Generate a random population $P_{0;}$
2. Sort population according to the non-domination and assign to each individual a fitness equal with its non-domination level. Level 1 is the best, level 2 second best and so on;
3. Perform selection, crossover and mutation to obtain a population Q_0 of size N;
4. Set generation counter t=0;
5. If generation have reached their maximum jump to step 6;
   5.1. Combine parent and offspring population $R_t = P_t \cup Q_t$;
   5.2. Find out all the non-dominate fronts $F = (F_1, F_2, \ldots)$ of R_t using the non-dominated sorting algorithm;
   5.3. The new parent population $P_{t+1} = \phi$ and counter *i=0;*
   5.4. If P_{t+1} is full ($|P_{t+1}| + |F_i| > N$)jump to step 5.5;
      5.4.1. compute the crowding distance of the current front $F_{i;}$
      5.4.2. the i^{th} nondominated front is inserted in the new parent population $P_{t+1} = P_{t+1} \cup F_i$. Increase the counter *i = i+1;*
      5.4.3. jump to step 5.4;
   5.5. At this point parent population P_{t+1} has fewer than N members (see condition at step 5.4). $N - |P_{t+1}|$ members have to be selected from the last front F_i. F_i is sorted using $\prec_n$. After this sorting procedure the first $N - |P_{t+1}|$ from F_i are selected and added to P_{t+1};
   5.6. use selection, crossover and mutation to create a new population $Q_{t+1;}$
   5.7. increment generation counter t = t+1;
   5.8. jump to step 5;
6. STOP.

2.3.3 SPEA2

Strength Pareto Evolutionary Algorithm 2 (SPEA2) [14] is a development of SPEA [15].

In contrast with NSGA-II, SPEA2 uses an external archive which keeps the best found individuals. Besides this, the fitness assignment method is different from the one used by NSGA-II. These are the two main differences. We will further present the algorithm.

Before running the algorithm some parameters have to be specified: N = population size; $\overline{N}$ = archive size; T maximum number of generations.

The algorithm:

1. Generate the initial population P_0 and create an empty archive $\overline{P_o} = \theta$. Set the generation count t = 0;
2. If stop condition has been met jump to step 3;
   2.1. Unite the current population P_t and the archive $\overline{P_t}$ into a single population (union)
   2.2. Compute the fitness of individuals in the union (presented below);
   2.3. The nondominated individuals from the union are copied in the archive $\overline{P_{t+1}}$;
   2.4. If the size of $\overline{P_{t+1}}$ is greater than $\overline{N}$ then eliminate individuals using the truncation method (presented below);
   2.5. If the size of $\overline{P_{t+1}}$ is less than $\overline{N}$ then copy the best dominated individuals from the union to the new archive (presented below);
   2.6. Perform selection on $\overline{P_{t+1}}$;
   2.7. Perform crossover and mutation on the selected individuals and obtain the offspring population (P_{t+1});
   2.8. t =t+1;
   2.9. Go to step 2;
3. STOP.

Fitness assignment (step 2.2)

The fitness assignment method was developed to avoid situations where individuals dominated by the same archive members have the same fitness value. To do this the method takes into consideration both the nondominated individuals and the dominated ones. For each individual i from the union (P_t and $\overline{P_t}$) the strength $S(i)$ is computed. $S(i)$ is equal with the number of individuals that the current individual dominates:

$$S(i) = \left| \left\{ j \,\middle|\, j \in P_t \cup \overline{P_t} \wedge i \succ j \right\} \right|$$

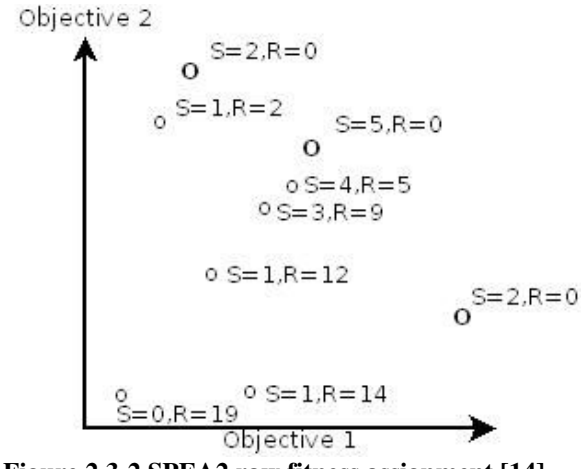


Figure 2.3-2 SPEA2 raw fitness assignment [14]

Using $S(i)$ a *raw fitness* value $R(i)$ is computed. $R(i)$ represents the sum of the strengths of the individuals that dominate individual i:

$$R(i) = \sum_{j \in P_t \cup P_{t+1},\, j \succ i} S(j)$$

The fitness needs to be minimized. If $R(i)$ = 0 then the individual is nondominated. If an individual is better, then its strength will be high and its raw fitness will

be low. An example with the raw fitness assignment is presented in Figure 2.3-2 (the example is taken from [14]).

Evolutionary algorithms tend to lose diversity and thus converge into a single solution. To avoid this mechanisms are needed to maintain the diversity. The raw fitness assignment does this, but if the most individuals do not dominate each other, it might not be enough. The authors of SPEA2 use density information to separate the individuals with the same raw fitness value. They use an adaptation of the k-th nearest neighbor method [16]. In this algorithm, a density parameter is associated with each point. This density is a function of the distance between this point and the kth nearest point. First the distances from individual *i*, in the objective space, to all the other individuals are computed. Then the distances are sorted in increasing order. σ_i^k is the inverse of the distance to the kth nearest neighbor selected and is used to compute the density information:

$$D(i) = \frac{1}{\sigma_i^k + 2}$$

where *k* is computed as the square root of the sample size [16]: $k = \sqrt{N + \overline{N}}$. The two (2) is added to force $D(i)$ in the interval $(0,1)$.

To compute the fitness for individual *i* is the following equation is used: *F(i) = R(i)+D(i).* The nondominated individuals will have a fitness value between 0 and 1.

Environmental selection (step 2.4 and step 2.5)

In step 2.3 from the algorithm, all the individuals that are nondominated (fitness value between 0 and 1) are copied in the archive:

$$\overline{P_{t+1}} = \left\{ i \middle| i \in P_t \cup \overline{P_t} \wedge F(i) < 1 \right\}$$

If the number of individuals from $\overline{P_{t+1}}$ is exactly $\overline{N}$ then nothing extra has to be done. If it is not the case, then there are two situations:

1) $\left|\overline{P_{t+1}}\right| < \overline{N}$

In this situation the best dominated individuals from P_t and $\overline{P_t}$ are copied in $\overline{P_{t+1}}$ until the archive is full $\left|\overline{P_{t+1}}\right| = \overline{N}$

2) $\left|\overline{P_{t+1}}\right| > \overline{N}$

When this happens, a density information is used to discriminate between "equal" individuals (truncation method). The truncation method works like this: first it computes the distances between all the nondominated individuals. The individuals which have the smallest distance to their neighbors will be eliminated from the archive. An intuitive example of how this truncation method works is explained below (see also Figure 2.3-3). The example is taken from [14].

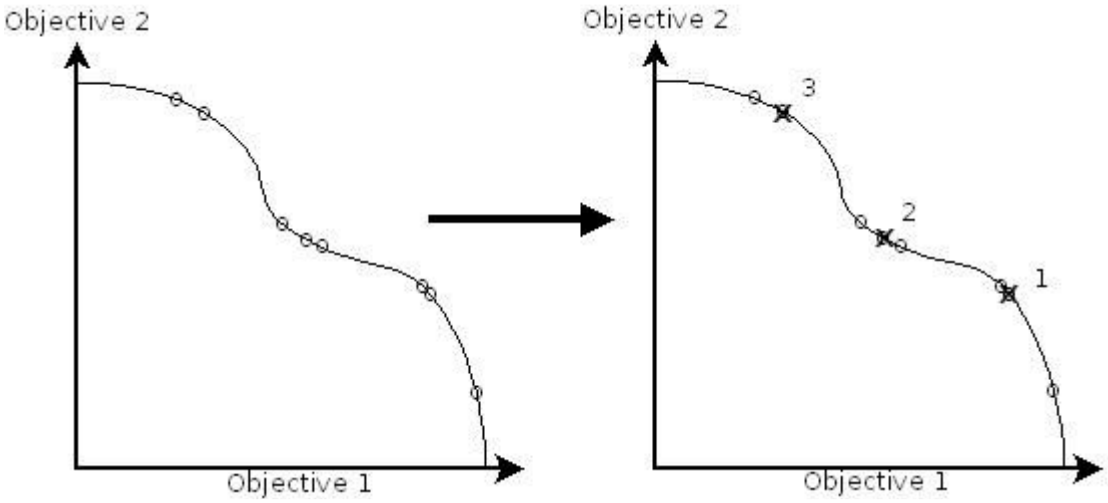


Figure 2.3-3 SPEA2 archive truncation method [14]

The first individuals to be found are the ones near the "1" zone. They are the closest. The one below is removed since it is closer to another individual. Next the individuals in the zone "2" are found. The one above is removed since it is closer to the individual above him. Next the individual in zone 3 is removed using the same algorithm. The advantage of this method is that the border individuals will not get removed (the borders are considered to be at an infinite distance).

2.3.4 Comparison between SEMO and FEMO

2.3.4.1 Synthetic Test Functions

We will present only the synthetic functions used in our experiments.

Leading Ones Trailing Zeroes (LOTZ) problem

The problem was proposed in [11] and it is a generalization to two dimensions of the "Leading Ones" problem.

The function LOTZ : $\{0,1\}^n : N^2$ problem is defined as:

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

The objective is to maximize the number of leading ones and the number of trailing zeroes in a bit-string simultaneously.

Since there is no point that maximizes both objectives simultaneously the multi-objective algorithm will need to find all the nondominated points.

DTLZ problem family

DTLZ1 test function was introduced in [17]. It is defined for any number of objectives and any number of decision variables. For an *m* objective problem with *n* decision variables it is defined as follows:

Minimize $\vec{f}(\vec{x})$

$$f_1(\vec{x}) = \frac{1}{2} x_1 x_2 ... x_{m-1} (1 + g(x_m, x_{m+1}, ..., x_n)),$$

$$f_2(\vec{x}) = \frac{1}{2} x_1 x_2 ... x_{m-2} (1 - x_{m-1})(1 + g(x_m, x_{m+1}, ..., x_n)),$$

$$f_3(\vec{x}) = \frac{1}{2} x_1 x_2 ... x_{m-3} (1 - x_{m-2})(1 + g(x_m, x_{m+1}, ..., x_n)),$$

…

$$f_{m-1}(\vec{x}) = \frac{1}{2} x_1 (1 - x_2)(1 + g(x_m, x_{m+1}, ..., x_n)),$$

$$f_m(\vec{x}) = \frac{1}{2} (1 - x_1)(1 + g(x_m, x_{m+1}, ..., x_n)).$$

subject to $0 \le x_i \le 1$ for $i = 1,2,...n.$

and $g(\overrightarrow{x_M}) = 100\left((n-m) + \sum_{i=m}^{n} (x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5)) \right)$

where $\overrightarrow{x_M} = (x_m, x_{m+1}, ..., x_n)$.

The Pareto optimal solutions correspond to the hyperplane that intersects the coordinate axes in the objective space at the value 0.5.

2.3.4.2 Results Comparison

In [18] we have performed a comparison between SEMO and FEMO (similar with [11]).

Both SEMO and FEMO start from a single individual. We observed that the position in space, where this initial individual is placed, is essential to the quality of the results. To avoid biased results we have run all the simulations 1000 times, each time starting from a random individual, and computed the average.

We tested SEMO and FEMO on two synthetic test problems: LOTZ and DTLZ1. The advantage of a synthetic problem is that the true Pareto front is known and metrics like error ratio (see Paragraph 2.6.2.1) can be used.

For the LOTZ problem, the Pareto front is finite, so both algorithms were run until they found all the Pareto optimal solutions. We configured it with 16 decision variables, thus an exhaustive evaluation requires 65536 (2^{16}) evaluations. SEMO discovers the true Pareto front in an average of 1453 generations (around 2% of the whole effort). FEMO required, to discover all the Pareto optimal solutions, only 756 generations in average (around 1% of the effort).

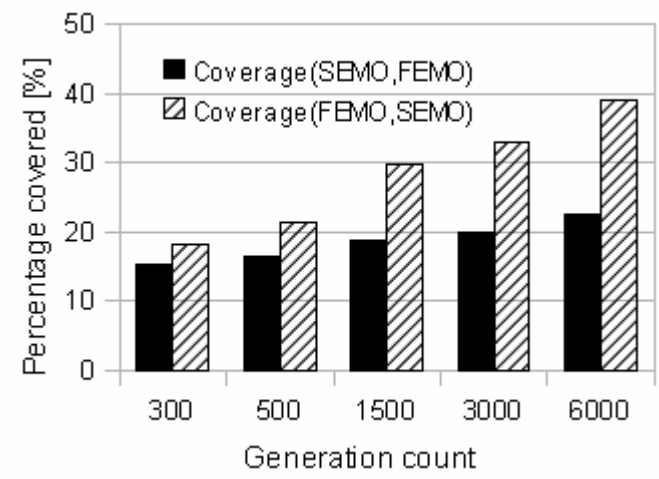


Figure 2.3-4 Coverage comparison on the DTLZ1 problem

The true Pareto front for the DTLZ1 problem is a hyperplane. This means that the number of optimal solutions is theoretically infinite, which is the reason the above test was not performed is. For this test problem we have computed the coverage metric (see Paragraph 2.6.3.4) at the end of several runs. DTLZ was configured to

have 7 decision variables and 3 objectives. The runs were stopped after 300, 500, 1500, 3000 and 6000 generations and coverage was compared (see Figure 2.3-4). It can be seen that FEMO performs better than SEMO for this problem too.

We analyzed the approximated Pareto front obtained by the two algorithms. Even after 12000 generations the true Pareto front of the DTLZ1 problem was not reached. The explanation for this is that both algorithms generate a single individual per generation, plus they lack many features of the modern evolutionary algorithms. In [17] it is reported that NSGA-II reaches the true Pareto front in 300 generations with a population of 100 (so 30000 evaluations).

2.3.5 Comparison between NSGA-II and SPEA2

It is interesting to see the difference between NSGA-II and SPEA2 in terms of fitness assignment. The question is what happens if there are two identical individuals in the population? In SPEA2 both individuals will have the same fitness assigned: they dominate/ are dominated by the same individuals, thus the value of the raw fitness is identical. When the density information is computed, the same number is obtained for both individuals (they have the same k^{th} nearest neighbor).

In NSGA-II these two identical individuals will have a different fitness value assigned. During the front extracting procedure they will end up in the same front and the same fitness will be assigned. The difference appears during the density computation. The individuals are sorted according to one objective. Each individual is compared with two of its neighbors. The first individual will have to his "left" (sorted vector) an individual and its twin to the "right". The second individual will have to the "left" its twin but to his "right" a different individual, thus the rectangles formed by their neighbors have a great chance of being different. As a result the final fitness will have different values.

This is important, as we demonstrate in Chapter 5.5, in terms of number of unique individuals found in the population. SPEA2 tends to have more duplicates than NSGA-II in the final population. 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. If there are fewer nondominated individuals than the maximum size of the archive/population, 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. More details will be provided in Chapter 5.5.

2.4 Multi-objective Particle Swarm Optimization

The most important problem faced by the multi-objective PSO algorithms is that multiple leaders can exist at the same time. The simplest approach is to select randomly one of them.

2.4.1 OMOPSO

OMOPSO [19] is a particle swarm optimization method. The authors do not clearly explain the acronym but it seems to come from Our Multi-Objective Particle Swarm Optimization. It uses only the global best and the personal best information. For understandability we repeat the equations of the PSO algorithms. The speed is updated according to the following formula:

$$\vec{v}_i(t) = W\vec{v}_i(t-1) + C_1 r_1(\vec{x}_{pbest_i} - \vec{x}_i(t-1)) + C_2 r_2(\vec{x}_{gbest} - \vec{x}_i(t-1))$$

With this value the individual updates his position (mutation with conscience):

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t)$$

where W is the inertia weight, C_1 and C_2 are the learning factors (usually constants – but not in this work) and r_1, r_2 are random values in the interval [0, 1]. The authors argue that is very hard to set the W, C_1 and C_2 parameters. In this article W is set to a random value between 0.1 and 0.5, C_1 and C_2 to random values between 1.5 and 2.0. With this approach each particle will be computed with different parameters. The values for the limits of the random functions are selected from their own experience/ previous work.

The problem with multi-objective optimization and particle swarm optimization is how to choose a leader when multiple solutions are "the best" (nondominated). One solution is to consider all the nondominated solutions as leaders. A problem that can arise is that the number of leaders increases over time if this method is applied. Thus two problems have to be solved: (1) how to choose a leader from the nondominated front and (2) how to limit the number of nondominated individuals (leaders).

First density information is added to the leaders. The same technique is used like the one in NSGA-II. The leader is then chosen using binary tournament selection like in NSGA-II based on this density information. To limit the number of leaders at each generation this density is computed and if there are too many leaders (bigger than the swarm size/ population size) the most crowded ones are eliminated.

OMOPSO uses the concept of *e*-dominance [20] to keep an archive of the leaders.

Definition: A decision vector x_1 is said to *e*-dominate a decision vector x_2 (in a minimization problem) for some $e > 0$ if: $f_i(x_1)/(1+e) \leq f_i(x_2)$, $\forall$ i=1,…,m and $f_i(x_1)/(1+e) < f_i(x_2)$, for at least one i=1, …, m. This means that an individual, to dominate another individual, has to be at leas at some distance from the other one. If it is very close they will be considered nondominated.

OMOPSO uses three structures: the swarm where all the particles are stored, the leaders were the nondominated solutions are kept and an external archive where individuals that dominate according to the *e*-dominance operator are kept.

Besides the turbulence operator the OMOPSO algorithm uses mutation. The authors use two types of mutation: uniform mutation (the maximum amount a variable might change is constant over time) and non-uniform mutation (the maximum amount a variable might change decreases over time, see Chapter 2.1.3.1). In classic PSO algorithms all the individuals are changed when the turbulence operator is applied. In OMOPSO, after the particles are "flown" through space, the swarm is divided into three. The first set of individuals will remain the same – no mutation applied. The second set will be modified using uniform mutation; while for the third set non-uniform mutation is used. The leaders are common to all the subsets.

The algorithm works as follows (jMetal library implementation):

1. Initialize swarm (randomly);
2. Select the leaders from the swarm, using the dominance relation;
3. Add leaders to the *e*-archive, according to the *e*-dominance;
4. While the maximum generation has not been reached do (if it has been reached jump to step 5);
   - 4.1. For each particle in the swarm;
     - 4.1.1. Select a leader;
     - 4.1.2. Compute speed for this particle taking into consideration its personal best and the selected leader as a global best (a different speed is

computed for each parameter of the individual – the parameters of the formula stay constant, they are randomized for each individual only);
4.1.3. Move (fly) the particle according with its speed;
4.1.4. Apply mutation (depending on the set it is);
4.1.5. Evaluate the particle after mutation;
4.1.6. Update p_{best} (personal best) – the personal best is updated if the new individual dominates the old personal best or if they are nondominated;
4.2. Update leaders (only the individuals who changed their personal best will be taken into consideration);
4.3. Send leaders to *e*-archive;
4.4. Compute the crowding of the leaders;
4.5. Jump to step 4;
5. STOP.

2.4.2 SMPSO

Speed-constrained Multi-objective PSO (SMPSO) [21] is developed from OMOPSO. The authors argue that they compared multiple PSO state of the art algorithms in a previous article [22] and found out that OMOPSO has the best overall performance. However, still OMOPSO is not able to solve some of the test problems (ZDT4 [23]). They conclude that this is because the speed (velocity) of the particles becomes very high at some points in the algorithm. This behavior is called "swarm explosion" [24] and can be solved by adopting a velocity constraint algorithm [24].

Since SMPSO is based on OMOPSO it is almost the same algorithm. There are some differences: it makes use of a constriction coefficient, the interval for the random values C_1 and C_2 is changed and others.

The constriction coefficient χ

$$\chi = \frac{2}{2 - \varphi - \sqrt{\varphi^2 - 4\varphi}}$$

where

$$\varphi = \begin{cases} C_1 + C_2, C_1 + C_2 > 4 \\ 1, C_1 + C_2 \le 4 \end{cases}$$

The velocity is computed according to the usual formula and then multiplied with the constriction coefficient χ. The obtained speed for each variable *j* in each particle *i* is constrained using the following formula:

$$v_{i,j}(t) = \begin{cases} delta_j, v_{i,j}(t) > delta_j \\ -delta_j, v_{i,j}(t) \le -delta_j \\ v_{i,j}(t), otherwise \end{cases}$$

where

$$delta_j = \frac{(\text{upper\_limit}_j - \text{lower\_limit}_j)}{2}$$

In Figure 2.4-1 this effect is shown.

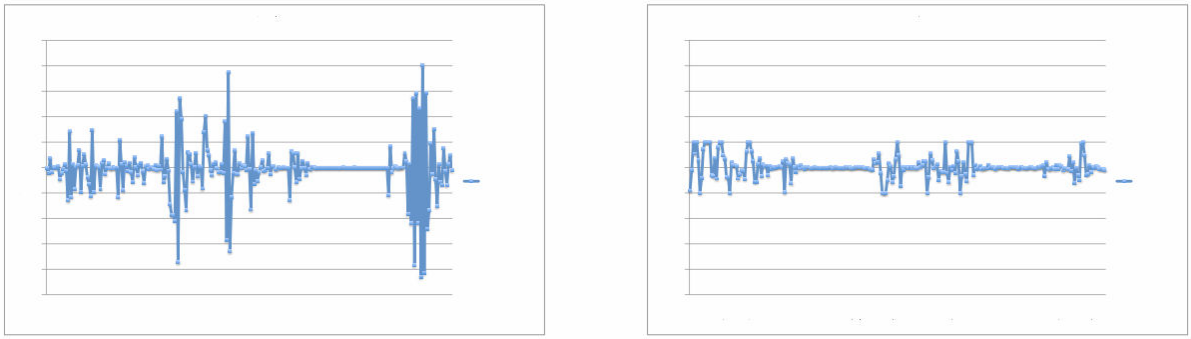


Figure 2.4-1 Velocity value of one of the parameters when solving ZDT4 for: OMOPSO (left) and SMPSO (right) - Figure extracted from the original article describing SMPSO [21]

Other changes to the OMOPSO algorithm

The range in which C_1 and C_2 are varied is changed in SMPSO. In OMOPSO the range was [1.5, 2.0]. This would mean that φ would be always 1 (sum of C_1 and C_2 will always be smaller or equal with 4). The authors changed this range to [1.5, 2.5].

In OMOPSO, after the parameter is updated, it checked if the limits of the parameter had been crossed. If this happens the parameter is set to its lower or upper boundary (depending on which was crossed). In addition in OMOPSO the velocity is multiplied by -1 in this situation. In SMPSO instead of multiplying the velocity with -1 they multiply it with 0.001. The authors argue that in their experiments this provided better results.

As we already presented in OMOPSO, the swarm was divided into three categories. In SMPSO only one mutation operator is used (polynomial mutation) and it is applied to 15% of the particles with a probability of 1/number of variables.

Almost all of our explorations (except the synthetic ones) use integer parameters. Both OMOSPSO and SMPSO were designed to work with float values. We changed both algorithms to work with integer parameters. The values were rounded just before assigning them to the parameters. Some of the articles that use and recommend this method are: [25] and [26].

2.5 Handling Constraints

The method we used for constraints handling is proposed in [13] and is employed in couple with binary tournament selection.

When constrains are present there are three possible situations that can occur during a binary tournament selection: both individuals are feasible, one is feasible and the other is infeasible and both are infeasible. In unconstrained multi-objective problems, that use binary tournament selection, the individuals are compared using the domination relationship. The domination relationship needs the value for each objective so it can perform the comparison. For an infeasible individual these values can not be obtained because the individual can not be simulated. 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.

It is interesting to see that in the last situation an infeasible individual is still accepted in the population. We will explain why this is good in certain situations with a simple example.

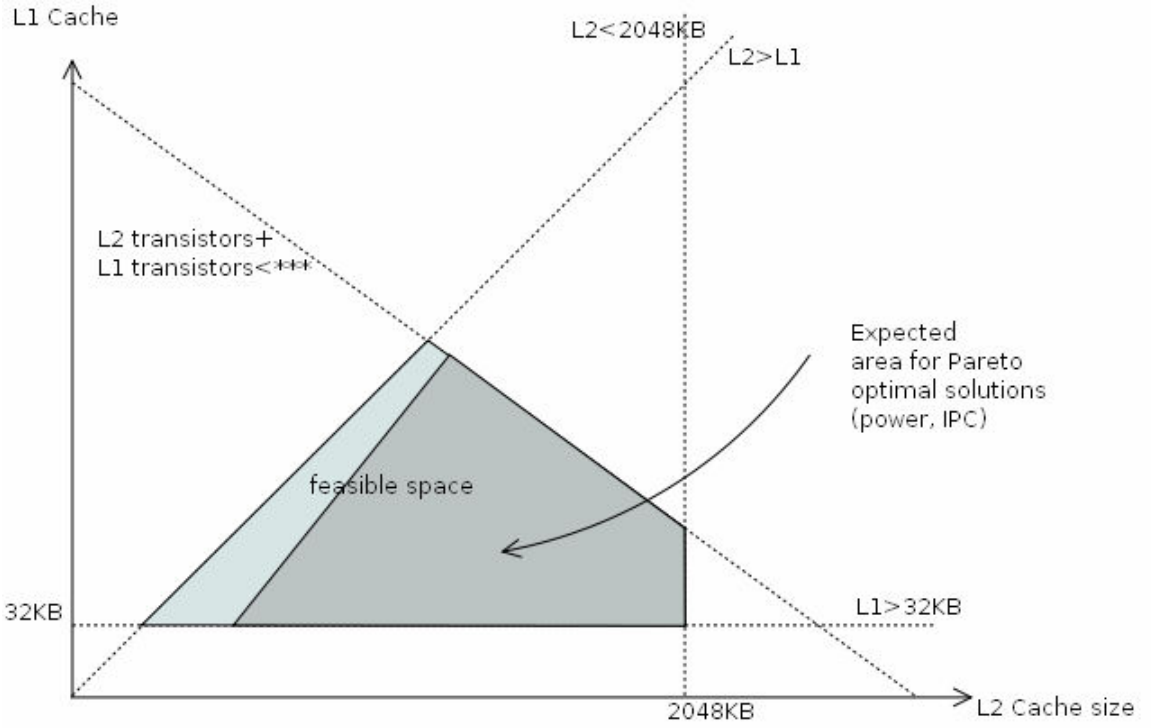


Figure 2.5-1 Constrained space

In Figure 2.5-1 we have defined the following constraints in the parameter space (we have only two parameters):

- Level 2 cache size must be smaller than 2048 KB;
- Level 2 cache size must be bigger than level 1 cache size;
- Level 1 cache size must be bigger than 32KB;
- And the numbers of transistors used for both cache must not exceed a certain limit (this is just an example).

These constraints limit the design space. The area where we will most likely find very good configurations in terms of performance is marked on the figure (the regions are where the caches are big). At the extreme, the best performance (we are not taking into consideration other objectives here like power consumption, integration area) should be obtained on the borders (L2 cache 2MB).

If infeasible individuals are not accepted in the population, the algorithm tends to not explore the borders, where in fact very good individuals reside. It has been observed that allowing individuals that do not respect the constraints provides better results. They might carry information which is valuable, or they might have crossed the border just a bit and by a mutation operation can become feasible again.

2.6 Measuring Multi-objective Algorithms Performance

2.6.1 Classification

We can divide the metrics in Pareto compliant and Pareto noncompliant methods.

An indicator $I : \Omega \to R$ is Pareto compliant if for all $A, B \in \Omega : A \preceq B \Rightarrow I(A) \geq I(B)$, assuming that greater indicator values correspond to higher quality (otherwise $A, B \in \Omega : A \preceq B \Rightarrow I(A) \leq I(B)$).

Pareto noncompliant is any indicator that can yield for any approximation sets $A, B \in \Omega$ a preference for A over B, when B is preferable to A with respect to weak Pareto dominance.

Further we divide the methods in two classes: that require the Pareto front to be known and the ones that don't. In real life problems the Pareto front is usually not known. In this thesis we will be using the ones that do not require the Pareto front.

2.6.2 Metrics That Require Pareto Front to Be Known

2.6.2.1 Error Ratio

The Error Ratio (ER) [20] counts the individuals in the known Pareto front that are not members of the true Pareto front. This metric is Pareto compliant.

Mathematically:

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

where $|PF_{known}|$ is the number of points in the known Pareto front and $e_i = 0$ when the i^{th} individual from the known Pareto front is an element of the true Pareto front (or is not dominated by any point from the true Pareto front). $e_i = 1$ if the individual from the known Pareto front is **not** an element of the true Pareto front. If ER is 0 then the known Pareto front is the true Pareto front. If ER is 1 then none of the elements from the known Pareto front belong to the true Pareto front.

2.6.2.2 Generational Distance

The Generational Distance (GD) reports how far, on average, the known Pareto front is from the true Pareto front. This metric is Pareto noncompliant.

$$GD = \frac{\left(\sum_{i=1}^{n} d_i^p\right)^{1/p}}{|PF_{known}|}$$

$|PF_{known}|$ is the number of points in the known Pareto front. p is usually 2 and d_i is the Euclidian distance between an individual from the known Pareto front and the closest point from the true Pareto front. If GD = 0 then the known Pareto front is the true Pareto front.

2.6.2.3 Hypervolume Ratio

It is based on the metric "size of space covered" (see metrics when the Pareto front is not known). The size of space covered measures the volume enclosed in the objective space by the known solutions. The hypervolume ratio computes the ratio between volume enclosed by the known Pareto front and the volume enclosed by the true Pareto front:

$$HR = \frac{S_{known}}{S_{true}}$$

This metric is Pareto compliant.

2.6.2.4 Average Distance from Reference Set

To measure the distance between the true Pareto front and the approximated front we can use the Average Distance from Reference Set (ADRS) [27] [28]. Let us assume A $\in$ Ω is a set of individuals, while p(A) is the non-dominated set, the function ADRS measures the distance (in the objective space) between p(A) and the Pareto-optimal set X p = p(Ω):

$$ADRS(X_p, p(A)) = \frac{1}{|X_p|} \sum_{\vec{x_p} \in X_p} \left(\min_{a \in p(A)} \left\{ d(\vec{x_p}, \vec{a}) \right\} \right)$$

where:

$d(\vec{x_p}, \vec{a}) = \max_{j=1,\ldots,m} \left\{ 0, \frac{f_j(\vec{a}) - f_j(\vec{x_p})}{f_j(\vec{x_p})} \right\}$ and *m* is the number of objectives

2.6.3 Metrics That Do Not Require Pareto Front to Be Known

2.6.3.1 "7-Point" Average Distance Measure

The "7-Point" average distance measure is similar to generational distance. For this metric the true Pareto front is not needed. It generates seven points (vectors) in objective space for comparison.

Assuming a bi-objective maximization problem and an (f1, f2) coordinate system with origin at (0,0). The maximum value in each objective dimension is found (f_{1max} and f_{2max}). The first two points from the total of seven are: (0,f_{2max}) and (f_{1max},0). The next five points are: (0,f_{2max}/3), (0,f_{2max}/2), (f_{1max}/3,0), (f_{1max}/2,0) and (0,0) as shown in Figure 2.6-1.

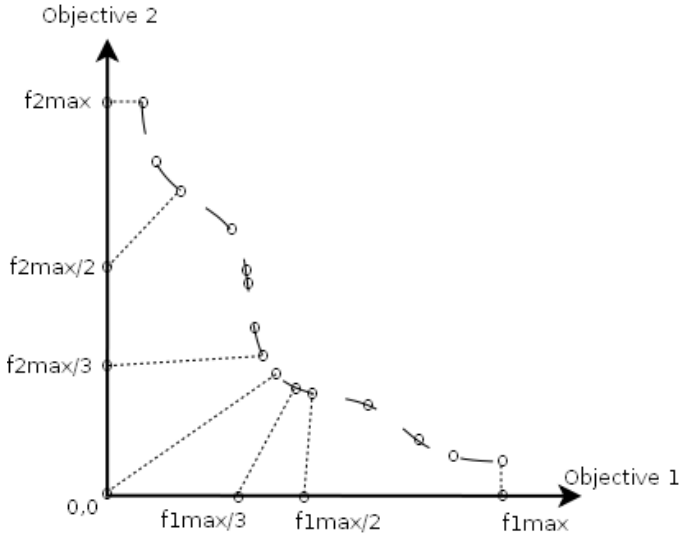


Figure 2.6-1 "7 Point" average distance

The "7-Point" average distance measure is the average Euclidean distances from each of the seven axis points to closest individual from the Pareto front approximation.

2.6.3.2 Hypervolume

This metric was used by Zietzler and Thiele [15] and Coelo [20]. 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. In the two-dimensional case $S(X')$ represents the area of the reunion of all the rectangles defined by an individuals and the (0,0) coordinate in the objective space.

This method has the advantage that each algorithm can be compared independently of another.

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.

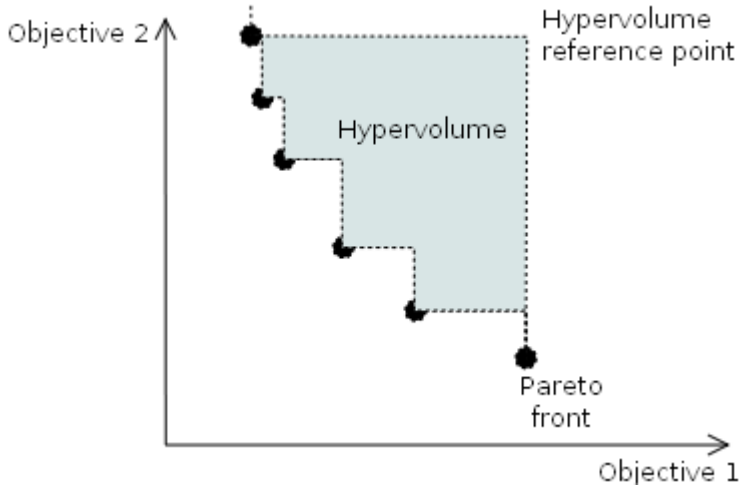


Figure 2.6-2 Hypervolume computation for a minimization problem

The hypervolume reference point coordinates are set to the maximum values of the objectives (see Figure 2.6-2). 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). This metric is Pareto compliant.

2.6.3.3 Two Set Difference Hypervolume

This metric was proposed by E. Zitzler in his PhD thesis [29]. 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''.

2.6.3.4 Coverage of Two Sets

This metric was used by Zietzler and Thiele [15] and Palermo [27].

It is used to compare different algorithms or different runs of the same algorithm. 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]:

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

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

2.7 Summary

In this chapter we proposed some classification methods, presented the algorithms that we are using and other information necessary to understand the work we have done.

An introduction to multi-objective optimization has been performed. We focused on Pareto based methods. Multi-objective genetic algorithms have been presented and compared. From the comparison of SEMO and FEMO on synthetic problems we concluded that the added knowledge indeed helps FEMO perform better. The second comparison was done between NSGA-II and SPEA2. From this comparison we can expect that SPEA2 will produce less unique individuals, and will have more duplicates in the archive, leading to a worse spread of solutions.

Then we focused on the bio-inspired algorithms and analyzed two algorithms: OMOPSO and SMPSO. SMPSO is a development of OMOPSO and we analyzed the causes that lead to this new algorithm.

A constraint handling method commonly used in optimization problems is presented. This is mandatory since most of real world problems (ours too) are constrained.

In the end of the chapter we have focused on the metrics used to measure the performance of the algorithms and the quality of the solutions they provide. We stressed out that we need metrics that do not require the true Pareto front to be known, because it is not available in real world problems. From all the presented metrics we want to reiterate two of them which are used extensively throughout this work: hypervolume and coverage. Hypervolume measure the volume enclosed between the obtained Pareto front approximation and the axes in a maximization problem (a point called hypervolume reference point is selected for minimization problem). Coverage measures how many of the individuals from one population are dominated by individuals from another one.

More details about the algorithms can be found in our technical report "An overview of the multi-objective optimization methods" [30].

"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 evolutionary 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 [31]). All these features will be detailed in the next chapters.

FADSE integrates the jMetal library [32] which provides many state-of-the-art multiobjective evolutionary algorithms. Some of the implemented algorithms are: NSGA-II [13], SPEA2 [14], PAES [33], PESA-II [34], OMOPSO [19], MOCell [35] [36][37], AbYSS [37], MOEA/D [38], Densea, CellDE, GDE3 [39], FastPGA [40], IBEA [41], SMPSO [21]. jMetal includes optimization problems like: ZDT [23], DTLZ [42], WFG, CEC2009 and LZ09 families [38], Kursawe [43], Fonseca [44] and Schaffer classical problems and Srinivas, Tanaka, Osyczka2, Constr_Ex, Golinski and Water for the constrained problems class. It includes some quality indicators: hypervolume, spread, generational distance, inverted generational distance, additive epsilon. They require the true Pareto front to be known.

We have extended jMetal and we are able to perform design space exploration with simulators like: Multi2Sim [45], GAP [46], 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. In order to run it, a Java Runtime Environment (JRE) is needed which can be obtained at http://www.java.com/ . FADSE was tested with Java version 6.
The latest version of FADSE can be downloaded from http://code.google.com/p/fadse/. The source can be obtained using an SVN client like TortoiseSVN (http://tortoisesvn.tigris.org/). After installing a SVN client the source code can be downloaded using the following command:

svn checkout http://fadse.googlecode.com/svn/trunk/ fadse-read-only

A precompiled version of FADSE can be obtained from here:

http://code.google.com/p/fadse/downloads/list

This does not need SVN installed, but it is usually an older version of FADSE.

3.1 Related work

3.1.1 ArchExplorer

Archexplorer [47] is an online website which allows system architects to upload their design of a system component (e.g. cache simulator). This new component is automatically integrated in the system and compared against other similar components through a design space exploration.

The algorithm used in Archexplorer is not a well known, it is described as a statistical exploration. The authors use operators borrowed from genetic algorithms (mutation, crossover, selection) but apply them in a slightly different manner.

The authors of Archexplorer divide an individual into genes (modules). Each gene is then divided into sub-genes (parameters of that module). Combinations of these genes have associated a probability of selection. At the beginning all the combinations have the same probability. As different combinations are evaluated, the best performing ones will have a greater chance of being selected. On the selected combinations mutation is applied. Mutation is first performed at the gene level (change the module) and after that at the sub-gene level (then changes its parameters). If new modules are uploaded on the site, the mutation will be biased towards them (these new modules will be included in the individuals). Crossover is performed in a slightly different manner than in genetic algorithms. The method uses only one "parent", the selected individual. From all the possible combinations, randomly one is selected. From that one, randomly are selected a mix of modules and parameters. The selected modules and parameters are inserted in the "parent".

Usually, when designing a new computer system there is an area constraint. In Archexplorer the combinations are sorted into buckets and each bucket contains combinations that fit into a given area interval. Then only the bucket that obeys the area constraint is used for exploration (or smaller ones).

3.1.2 M3Explorer

M3Explorer [31] is a tool quite similar with our developed FADSE. We developed FADSE because we wanted to have full control over the source and the implementation and because M3Explorer did not fully accomplish our requirements. M3Explorer is configured, like FADSE, through an eXtensible Markup Language (XML) interface, which is designed to connect to almost any existing simulator. The difference here is that the simulator has to adapt itself to the exploration tool, while in FADSE an internal connector has to be written (the DSE tool adapts to the simulator). M3Explorer includes some DSE algorithms, NSGA-II is the most well known and probably one of the most advanced from this framework. The authors also implement a method to accelerate the design space exploration process using response surface models, but the details about the implementation and usage are scarce [31]. A major drawback of M3Explorer is the lack of distributed evaluation.

3.1.3 Other Tools

There are many other tools for design space exploration. An interesting one is NASA [48]. The idea in NASA is to provide a simple to use interface so that the designer can easily integrate his/hers own DSE algorithms as well as the desired simulator (an objective we have also achieved with FADSE).

Magellan [49] is a framework that includes many search algorithms, from exhaustive search to genetic algorithms. It has some drawbacks: it is tied to a single

simulator (SMTSIM), it contains only single objective algorithms and it does not allow distributed evaluation of the design points.

There are many DSE frameworks, many oriented to specific domains. Some of them are: Metropolis [50], MILAN [51], EXPO [52], FAMA [53], SPLOT [54].

3.2 General Workflow

In this chapter, FADSE it is introduced in more detail. In Figure 3.2-1, the general structure of our tool is depicted. FADSE is configured using an XML file (see Chapter 3.6 for a detailed overview). From this file, our tool finds out the parameters, the simulator configuration and other information. We have a component in FADSE, called the *AlgorithmRunner,* which finds out what DSE algorithm has to be run (specified in the XML file) and loads the appropriate class from the jMetal library. The DSE algorithm starts generating "*Solutions*" (this is the term used inside jMetal for individuals). *Solutions* are sent for evaluation. To be able to solve a problem using jMetal the Problem interface has to be implemented. This is what the *SimulatorWrapper* does: (1) it converts the Solution objects to *Individual* objects and then (2) calls an abstract method: *performSimulation*.

(1) – the conversion adds extra information to the solution: name for parameters, path for simulator, etc. The new object is called an individual in FADSE;

(2) – to run a simulator, a connector has to be implemented (see Chapter 3.5). All the connectors must extend the *SimulatorWrapper* and implement the *performSimulation* method.

The correct simulator connector (specified in the XML interface) is loaded, which then starts the simulator. When the simulation is finished, an output reader class, implemented particularly for each simulator, parses the results and sends back the values for the objectives. The process is then restarted.

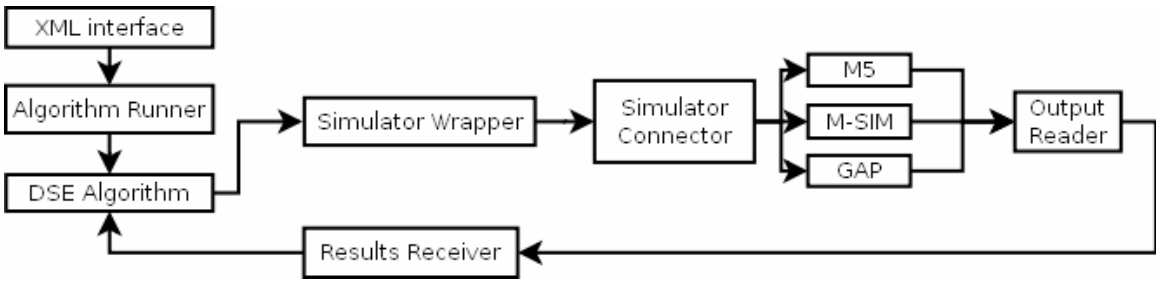


Figure 3.2-1 FADSE general structure

To make things more clear a simple example is depicted in Figure 3.2-2. In this example there is a single parameter, l1_cache_size, which is varied between values 32 and 512 using a geometrical progression with a ratio of 2 ("exp2" in the figure). The DSE algorithm generates a *Solution* with the parameter set to 64. For the algorithm, it is not important the name of the parameter, only its upper and lower bounds. The parameter is sent to the *SimulatorWrapper* which concatenates the name to the parameter. Afterwards the *Individual* is sent to the simulator connector which also obtains the path to the simulator. The *SimulatorConnector* builds the command line for the simulator and starts the simulation. The simulator produces an output file which contains the values of the objectives. Any type of output can be parsed because the parser is built specifically for a certain simulator. The simulator output is parsed and then sent back to the DSE algorithm. Again, for the DSE algorithm, it is not

important the meaning of the objectives (it uses objective 1 and objective 2 in the example). FADSE makes sure that this information is kept.

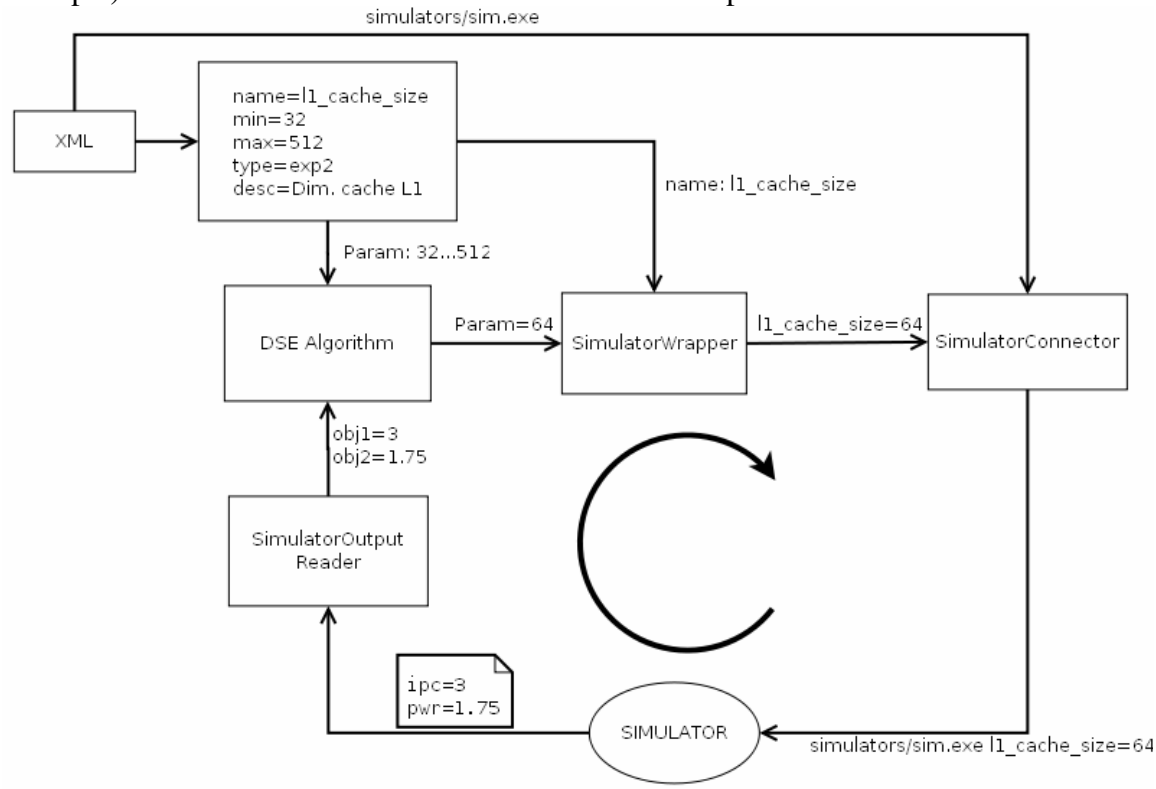


Figure 3.2-2 FADSE running instance

A design space exploration experiment takes a long time (we ran FADSE for over a month in some situations). A failure in the system is not acceptable. We have multiple mechanisms that assure the reliability of FADSE. They will be presented in the following chapters. In the following, we present one of them: checkpointing.

The checkpointing mechanism helps FASDE recover from the hardest to avoid errors/problems: power loss, bugs in the code, etc. At each generation, FADSE performs a checkpoint of the current state of the algorithm: individuals in the population/archive, variable parameters of the algorithm, etc. All the information required to restart FADSE from that point is saved. The checkpoint can be used for other purposes:

- FADSE can be started from an initial population given by the user;
- If multiple algorithms are compared they can all be started from the same initial population easily.

3.3 Accelerating DSE through Distributed Evaluation

In the previous chapter, we already stated that simulations might take a lot of time. For example with the M-SIM simulator it takes a day to evaluate a configuration on 12 benchmarks. Running a single generation of 100 individuals will take over 3 months. Running multiple generations is infeasible in this context. To avoid this problem we decided to parallelize FADSE and the DSE algorithms.

We observed that the DSE algorithms tend to evaluate an entire population and, only after all the individuals are ready, use these results to compute fitness, perform selection, etc. For a run with a population size of 100, all these individuals could be simulated in parallel. Even more, if an individual needs to be evaluated on

10 benchmarks it means 1000 simulations could be started in parallel, if the resources are available. We have distributed the following algorithms: AbBYS, Densea, FastPGA, IBEA, NSGA-II, OMOPSO, PESA2, SMPSO and SPEA2.

We will present the basic steps of many evolutionary/ bio-inspired algorithms:

(1) Generate initial population;
(2) Evaluate the initial population (set it as the parent population);
(3) Generate offspring from the parent population (mutation, crossover, turbulence operator, etc.);
(4) Evaluate offspring;
(5) Combine parent and offspring and extract the new parent population;
(6) If stop condition not reached jump to step (3);
(7) STOP.

The evaluations in step 2 (and also step 4) could be done in parallel; there is no interchanged information inside it. The only change to the algorithm will be a barrier inserted after each evaluation phase. The modified algorithm would look like this:

(1) Generate initial population;
(2) Evaluate the initial population (set it as the parent population);
Barrier;
(3) Generate offspring from the parent population (mutation, crossover, turbulence operator, etc.);
(4) Evaluate offspring;
Barrier;
(5) Combine parent and offspring and extract the new parent population;
(6) If stop condition not reached jump to step (3);
(7) STOP.

These two barriers raise some problems that FADSE has to solve. The evaluate (*performSimulation*) method has to be non-blocking. If the method does not release control back to the algorithm immediately after it is called, the algorithm will have to wait for the simulator to finish running and parallel execution will not be obtained. Besides this, FADSE had to implement some sort of communication protocol since we want to distribute the evaluations over a network.

We decided to use the client-server model. The server will run the DSE algorithm while the clients will be responsible with running the simulators.

To solve the problems stated above, we implemented a special simulator connector called *ServerSimulator*. This connector hides from FADSE the fact that it is running distributed. When a solution (individual) is sent for evaluation the *performSimulation* method is called by the *SimulatorWrapper* (see Figure 3.3-1). The *ServerSimulator* returns immediately a fake value saying that the simulation is finished (see Figure 3.3-1). In fact the *ServerSimulator* retains the *Solution* object and sends the converted to *Individual* solution to one of the clients. After the *ServerSimulator* obtains the objectives for this solution it injects back in the algorithm the values (he retained a pointer to the object).

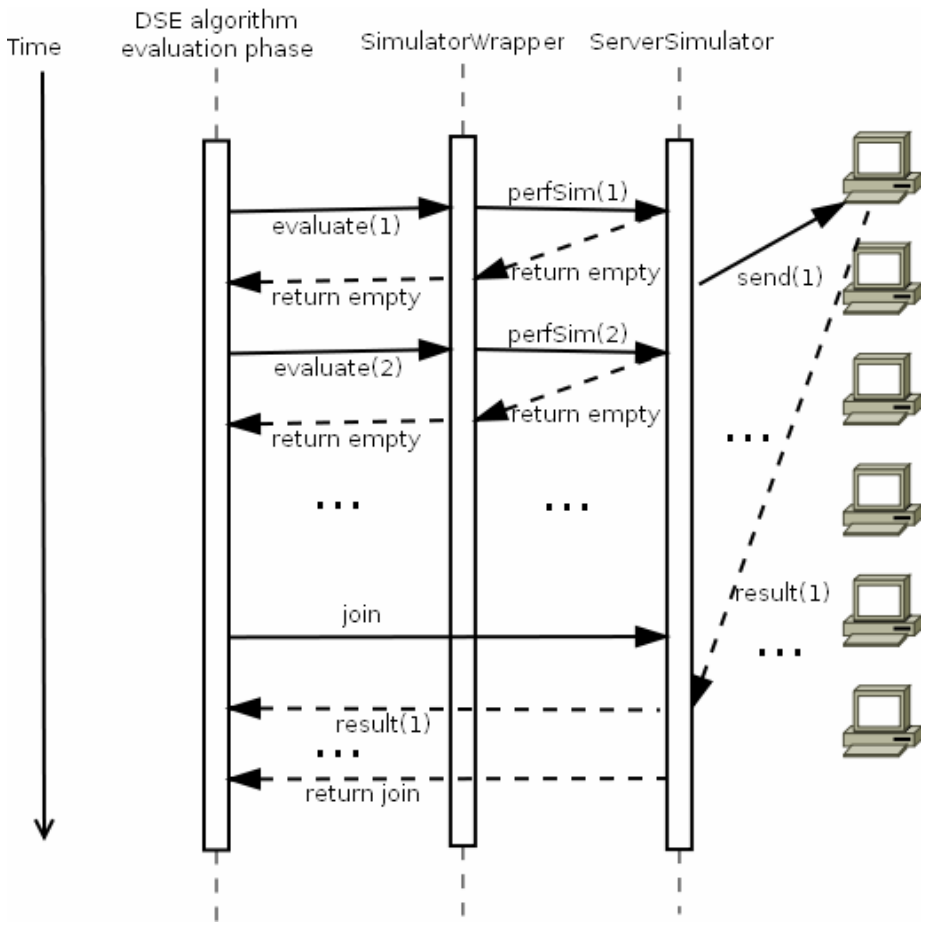


Figure 3.3-1 Masking the distributed evaluation from the DSE algorithm

The algorithm should not read the values of the objectives until they are written by the *ServerSimulator* (the barrier). To solve this problem the *ServerSimulator* implements a method called *join*. This is a blocking method and it is called by the algorithm after the evaluation phases. The *join* method does not release control until the *ServerSimulator* has results for all the solutions/individuals sent to simulation.

If multiple benchmarks are required for each individual the division is done by the *SimulatorWrapper*. It detects that there are several benchmarks, it clones the individual (one per each benchmark) and sends them for simulation (it does not matter if they are distributed or not on the network). Then it recovers the results and performs the average.

We must note that, when the *ServerSimulator* is used and multiple benchmarks need to be evaluated, some other problems arise. The *SimulatorWrapper* is unaware that it runs distributed so it sends the individuals to the simulator connector (*ServerSimulator*) and receives immediately a response and computes the averages, but in fact it does not have the final results. To solve this, the *ServerSimulator* computes the averages itself before injecting back the results into the solution object.

We decided that even from the simulator connector point of view there should be no difference if it runs in a client-server environment or not. On the client side we implemented the *IndividualReceiver* which receives the individual, loads the connector specified in the message and mimics the behavior of the *SimulatorWrapper*.

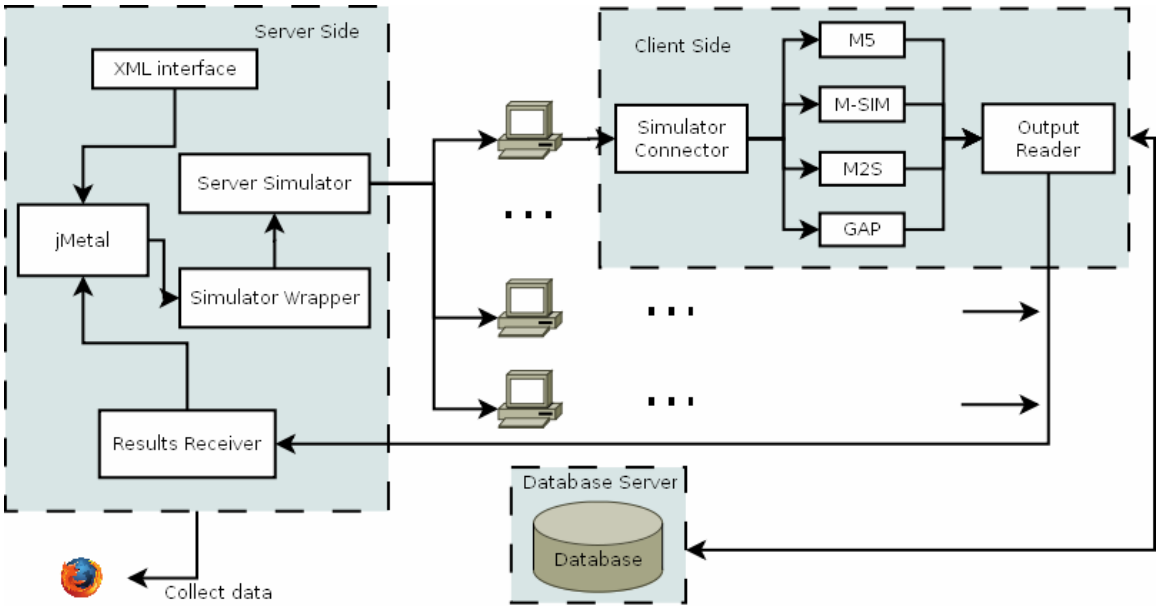


Figure 3.3-2 Structure of FADSE (distributed version)

In Figure 3.3-2 a simple representation of how FADSE works distributed is presented. The DSE algorithm and the connector side remain the same. The only difference is an added layer between the two which hides the distributed evaluation related problems.

Running in a distributed system increases the number of possible problems but FADSE includes multiple mechanisms to recover from these situations.

Some of the problems come from the network infrastructure, hanging clients and so on. To solve this, a watchdog timer has been implemented. This is a thread that monitors the client. If the client is not simulating and has not received a message from the server for a specified amount of time, the client is stopped and restarted. This means killing the entire process and the JAVA virtual machine and starting everything again. This helps recovering from various network problems and/or programming bugs (memory leaks).

Another reliability system is implemented on the server. The user has to specify the maximum time a simulation might last. If the server detects that a client has been simulating longer than this user defined time, it resends the individual to another client. The number of retries is limited and configurable from within FADSE. If the number of retries exceeded the limit, the individual is marked as infeasible[1]. In this situation we consider that the individual can not be simulated. In our experiments we allowed one retry. If the client, that should have responded, sends the results after the allowed period of time the result is saved by the server. When the *join* method is called if there are multiple results for the same individual the feasible ones are selected (the client marks an individual as feasible or infeasible).

To start the simulation using the client-server implementation, a special configuration file is required. This file contains the list of the names/IP and ports of the reachable clients to which FADSE can try to connect (if a client is specified but FADSE client is not started on that machine-port, FADSE skips it and moves to the next one). The file is called neighbourConfig.xml and has a structure like the one shown below:

[1] Infeasible means in this context that the simulator has crashed for this configuration

```
<?xml version="1.0" encoding="UTF-8"?>
</neighbors>
        <neighbor   ip="localhost" listenPort="4445" availableSlots = "1"/>
        …
</neighbors>
```

In this example, FADSE was configured to use a single client that runs on the local machine (the same machine where the server resides) on port 4445. Multiple clients can be started on the same machine with the restriction to listen on different ports. Any number of neighbors (clients) can be specified in this file. The *availableSlots* attribute specifies how many simulations can be run in parallel by a client. This feature is not implemented yet on the client side. The same behavior can be obtained by starting multiple clients on the same machine. This is useful when the machine is a multi-core/multi-processor.

It is important to know that the neighbor file can be modified while the DSE process is running. FADSE parses this file at the start of each generation and clients can be added or removed. This allows a flexible model to run the simulations. When resources are available FADSE can use more clients, but when they are required by another user, FADSE can be limited to a smaller number of clients without stopping the DSE process. In extreme situations, when resources are required by other users, the clients can be stopped even during a generation. FADSE will retry the simulations on other still running clients.

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 3.4 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 a batch system. 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.

3.4 Accelerating DSE through Results Reuse

DSE algorithms tend to produce the same individuals again after some generations. Instead of simulating them once more we could reuse the results from a database. For this we have connected FADSE with a Database Management System (DBMS). First we used Derby (http://db.apache.org/derby/) from the Apache foundation and then we switched to MySQL (http://www.mysql.com/) since we found it to be more versatile. We have implemented the database feature in a loose couple way, this makes switching to another DBMS (Data Base Management System) very easy. There is a single module that knows about the database structure/implementation, while FADSE works with only two methods. If these two methods are implemented (an insert and a search) any DBMS can be used.

The DBMS is invoked on the client side (see Figure 3.3-2). The client, before starting a simulation, searches the database if this individual has been simulated before or not. If it has not been simulated, the simulation is started and when it is finished the whole result is saved in the database. We are saving the whole result because it could be used in future DSE processes, when other objectives might be needed. Of course this can be changed by the connector developer. If the result is already in the database, then it is extracted and sent to the output parser. Form the point of view of the output parser it seems like the simulation has been run.

To make it simpler for the developers, we have implemented some helper classes which hide all the database logic. This class is called the SimulatorBase and it extends the *SimulatorWrapper* and implements the *performSimulation* method. More details about this in Chapter 3.5.

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.

The database is also used to avoid bugs in the code of the server. The *ServerSimulator* class has become very complicated because it has to track many concurrently running simulations. It has to track multiple simulations of the same individual (in cases of retries), search for unexpected behavior (e.g. objectives set to 0), compute the averages and select only the relevant individuals to do this. Bugs

might still appear in the code especially when retries are performed. To avoid this we are re-running all the simulations once a generation is done. This means that all the individuals are sent back to the clients, but this time it will be very fast because all the results are already in the database and no retries will be performed. The response arrives quickly back and with no problems. This way we can check if the DSE process runs correctly. The results are corrected automatically by FADSE.

3.5 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). For this, we analyzed many simulators (see Chapter 6.5 for some of them) and we have conclude that they usually can be configured using a command line and/or a configuration file. The simulators require parameters which most of the time have a name and a value. With this in mind we implemented the Individual class. This contains:

- the name of the parameters and their values;
- the benchmark that need to be run;
- the name of the objectives that need to be extracted;
- configurations for the simulator;
    - path to the simulator;
    - output file;
    - other user defined parameters.

The role of the connector is to aggregate all this data into a format accepted by the simulator, start the simulation with these parameters and at the end parse the results, extract the objectives and inject them in the *Individual* object.

So the *performSimulation* method has a parameter an instance of the *Individual* class, which needs to have its objectives filled when the method returns.

As we pointed out in the previous paragraph, some helper classes have been built to help the developer write a connector. The most important are: *SimulatorBase*, *SimulatorRunner* and *SimulatorOutputParser*. All these classes implement methods that are commonly used by all the connectors: database integration, reading files, etc. If the *SimulatorBase* is extended, for example, the developer implementing the connector does not need to know anything about the database, not even about its existence.

All the connectors written for FADSE extend these classes. Some of the connectors currently implemented are for: GAP and GAPtimize, M-SIM2 and M-SIM3, UniMap,M5, Multi2Sim.

Due to the small complexity of the FADSE-connector interface, a developer familiar with the simulator can implement a connector in a fairly small amount of time.

3.6 Extensible Input XML Interface

FADSE configures itself by reading the input XML file. This is mandatory for FADSE and has to be provided by the user when FADSE is started. The main structure of the input XML file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<design_space>
        <simulator name="…" type="…">…</simulator>
```

```
        <database ip="…" port="…" name="…" user="…" password="…"/>
        <benchmarks>…</benchmarks>
        <metaheuristic name="…" config_path="…" />
        <parameters>…</parameters>
        <system_metrics>…</system_metrics>
        <virtual_parameters>…</virtual_parameters>
        <rules>…</rules>
        <relations>…</relations>
    <design_space>
```

Each of these tags will be explained in more detail in the following chapters. The first tags are presented in this chapter while the *virtual_parameters*, *rules* and *relations* tags will be presented in Chapter 4.

3.6.1 Connector Configuration

FADSE supports running simulators or internal synthetic test problems (like the DTLZ family of functions). The user can choose between a simulator and a test problem from this tag.

For example the following construct selects a synthetic problem:

```
<simulator name="DTLZ1" type="synthetic"/>
```

If a synthetic problem is used, then none of the parameters, objectives, constraints (rules) specify below will have any effect. Since we have not used these synthetic problems extensively in this work we will no give further details.

A simulator can be specified in the following way:

```
<simulator name="Msim3Simulator" type="simulator" >…</simulator>
```

If a simulator is chosen, then parameters can be sent to the simulator connector. These parameters are not interpreted by FADSE and are only forwarded to the simulator connector written especially for this simulator. It is the responsibility of the user to know all the parameters that the simulator connector requires. They can be specified in the following way:

```
<simulator name=" Msim3Simulator " type="simulator" >
    <parameter name="simulator_executable" value="sim-outorder" />
    <parameter name="simulator_output_file" value="out#.txt"/>
</simulator>
```

Any number of parameters can be added by the user. These parameters are intended to configure the simulator connector. Usually paths to executables, output files are sent using this method. FADSE parses these parameters and searches for the "#" character. This character is replaced by a unique number. This is required if a user wants to keep all the output files, so it has to give them unique names. This feature is used when a distributed evaluation is run, because each client has to write in its own file. To start a distributed evaluation (besides the neighbor configuration file) the user has to add a small change in the input XML file:

```
<simulator name="ServerSimulator" type="simulator" >
        <parameter name="realSimulator" value="Msim3Simulator"/>
        <parameter name="simulator_executable" value="sim-outorder" />
        <parameter name="simulator_output_file" value="out#.txt"/>
</simulator>
```

Now the connector is the *ServerSimulator*. This requires a special parameter called *realSimulator* where the user has to specify the connector it wants to load on the client.

There are other three special parameters:

```
<parameter name="maximumTimeOfASimulation" value="820"/>
<parameter name="forceFeasibleFirstGeneration" value="true"/>
<parameter name="forceMinimumPercentageFeasibleIndividuals"
                value="80"/>
```

The first parameter specifies the number of minutes the server waits until it considers that the simulation/client has crashed and it has to resend the individual to another client. In this example we set it to 820 minutes, for a dual-core simulation on M-SIM 3.

The next parameter specifies if the first generation should be made entirely of feasible individuals. In Section 4.2 constraints will be introduced. The constraints are set by the user to instruct FADSE that some configurations are bad (infeasible[1]). The constraints are checked by FADSE for each individual. Setting this parameter to true will make FADSE randomly generate new individuals until it fills the entire initial population with good (feasible) individuals.

The last special parameter is *forceMinimumPercentageFeasibleIndividuals*. This is similar with the previous one but takes effect during the main loop of the algorithm. The heuristic algorithm will continue generating new individuals (crossover, mutation, etc.) until 80% of the population (in this example) is feasible. The rest 20% will be generated normally and it will not matter if they are feasible or not.

We introduced these parameters because, from our experiments, in some situations, we had many infeasible individuals and this reduced the number of good individuals that could survive to the next generation. This leads to a slower convergence speed.

3.6.2 Benchmarks

This following tag can be used when the user wants to perform simulations on a suite of benchmarks. The user can input a list of benchmarks. The *SimulatorWrapper* will receive this list and it will generate an individual for each benchmark and then send it to simulation. The *SimulatorWrapper* (*ServerSimulator* in a distributed run) returns an average value for the objectives. The benchmarks can be given to FADSE in the following way:

[1] The term of infeasible was used before in the context of crashed simulators. In this situation it refers to individuals that do not respect certain rules (constraints) imposed by the user.

```
<benchmarks>
        <item name="FFT"/>
        <item name="LU"/>
        <item name="Ocean"/>
</benchmarks>
```

3.6.3 Database Configuration

The *database* tag informs the framework about the location of the database.

```
<database ip="database_server" port="3306" name="fadse" user="fadse"
password="fadse"/>
```

In the tag the user has to specify the IP or the network name of the computer where the database server is located and the port on which the server listens. The *name* attribute specifies the name of the database. The *user* and *password* represent the username and the password required by FADSE to connect to the database.

3.6.4 DSE Algorithm Configuration

Through this tag the user can specify the desired design space exploration (DSE) algorithm. For example:

```
<metaheuristic name="NSGAII" config_path="nsgaii.properties" />
```

Some of the valid names for DSE algorithms are: AbYSS, CellDE, GDE3, MOCell, MOEAD, SMPSO, NSGAII, PAES, SPEA2, IBEA, OMOPSO.

Through the *config_path* attribute the DSE algorithm can be configured. If no file is specified, default values are used. In the configuration file, parameters such as: population size, mutation probability, maximum number of generations to run, etc. are given.

3.6.5 Architecture Parameters

In this section of the file, the parameters for the simulator are described. They describe the values an architectural parameter is allowed to take. For example:

```
<parameters>
        <parameter name="l1dcache_size" description="" type="exp2"
        min="32" max="128"/>
        <parameter name="nr_cores" description="" type="integer" min="1"
        max="4"/>
        <parameter name="branch_predictor" description="" type="string">
                <item value="GAg"/>
                <item value="Neural"/>
        </parameter>
</parameters>
```

In the above example we specify that the architecture we want to build should have a level 1 data cache of size between 32 and 128. For the level 1 data cache we have specified the type *exp2*. This means that the values produced will be a geometric

progression of ratio 2. In this case the values will be: 32, 64 and 128. We have also specified that we want to vary the number of cores between 1 and 4. The type is integer and the step is default set to 1. The values produced will be 1, 2, 3 and 4.

Then we have specified the types of predictors we want to vary. In the *branch_predictor* parameter we have specified the type string. This means that a list of one or many items is expected.

Next we will present each type of supported parameter in more detail.

3.6.5.1 Integer Parameters

<parameter name="…" description="…" type="integer" min="…" max="…" step="…"/>

This parameter type will produce an arithmetic progression between the values *min* and *max* with a ratio equal with *step*. The value of the *name* and *description* attributes can be any string. The *min* attribute specifies the minimum value that this parameter can have. The *max* attribute specifies the maximum value that this parameter can have. The *step* attribute is optional. If it is not specified its value is set to 1. With the step attribute the user can control the ratio of the arithmetic progression.

3.6.5.2 Exponential Parameters

<parameter name="…" description="…" type="exp2" min="…" max="…"/>

This parameter type will produce a geometric progression between the values *min* and *max* with a ratio equal with two. The value of the *name* and *description* attributes can be any string. The *min* attribute specifies the minimum value that this parameter can have. The *max* attribute specifies the maximum value that this parameter can have.

3.6.5.3 List of Strings

```
<parameter name="…" description="…" type="string">
        <item value="…"/>
        …
</parameter>
```

This parameter type will choose one of the values written in the item tags. The value of the *name* and *description* attributes can be any string. It can have one or many enclosed items. The *value* attribute of an item can be any string. Internally the strings are represented as integers starting from 0 to (number of items)-1.

3.6.6 Objectives

```
<system_metrics>
        <system_metric name="…" type="…" unit="…" desired="…"/>
        …
</system_metrics>
```

Within the *system_metrics* tag, the user has to specify the objectives of the problem. There can be specified any number of objectives. The type of the objectives can be only *float*. In the future we might use other type of objectives.

The *unit* attribute is used to specify the measurement unit. The measurement unit might be used by the *SimulatorConnector*. It has no other influence on FADSE.

The objectives can be minimized (desired="small") or maximized (desired="big"). If the attribute is not specified the default value is *small*. FADSE has been thoroughly tested on problems where the objectives had to be minimized.

3.7 Implemented Metrics

FADSE includes many metrics. Some of them are inherited from the jMetal library, but these metrics require the true Pareto front to be known. Since it is not available for real world complex problems, new metrics had to be implemented.

All the metrics used are able to read checkpoint files generated by FADSE. From the checkpoints, the individuals and their objectives are extracted to build the Pareto front approximation discovered until that point.

The first implemented metric was coverage (see 2.6.3.4). It can be used to compare two populations or two different runs. When two populations are compared, two checkpoint files have to be specified. When comparing different runs, the user has to specify the two folders where the checkpoints of the two runs reside. FADSE will then compute the coverage at each generation and generate a Microsoft Excel compatible file.

If a user wishes to observe the evolution of a single algorithm he/she can compute the hypervolume/ "7 point" average metric (see chapters 2.6.3.2 and 2.6.3.1). When the class, that computes these metrics, is ran FADSE also provides additional information about the DSE process. It computes: how many unique individuals were generated during the DSE process, how many from the produced offspring are unique, how many of them survive until the next generation (are accepted into the new parent population). It also generates a CSV file with all the names of the checkpoint files. This file can be used by scripts to generate images with the Pareto front approximations automatically. We have developed such scripts using the R tool (http://www.r-project.org/).

For the hypervolume we have also implemented a version that allows users to compare multiple runs. In this version, the maximums from all objectives, from all the runs are searched and this will be the hypervolume reference point, this way the runs will be comparable.

There is also an implementation for Hypervolume Two Set Difference (see Chapter 2.6.3.3) which is similar with the coverage metric in terms of usage.

All the metrics generate outputs in Microsoft Excel/Open Office compatible files. This way it is very easy for a user to generate charts.

3.8 Other Observations

In this paragraph other configuration files of FADSE are presented. The first one is *fadseConfig.ini*. This file is used to configure the server and client listening ports if the user wishes to run FADSE in a distributed manner. Below is the content of the file:

[Server]
ip = 192.168.1.102
listenPort = 4446
[Client]
listenPort = 4445
[Monitor]
listenPort = 4448
[RedistributeCheck]
timeSeconds = 30
[Watchdog]
time = 90

The values for the listening ports should not be changed, only if the ports are used on one of the machines. The IP address in the Server section is not used anymore and it will be removed in the following versions of FADSE.

The Monitor section configures a thread which monitors FADSE. This thread listens for outside requests and sends information about the status of the simulations ran by FADSE. This monitor is used so that an external web application can connect and gather information about the simulation progress. The monitor is still under development.

The next section in the INI file is *RedistributeCheck*. Here, the user has to specify how often FADSE has to check if a client has finished a simulation. For short simulations this should be a low value. If the simulations are long FADSE should not check very often (each check is logged so the log might get too big).

Under the *Watchdog* section the user has to specify how many minutes a client will wait for a message before it is restarted. If a client is not simulating, it will wait the specified amount of time and if it does not receive a message, it is restarted because it is assumed that something has gone wrong.

JMetal can configure its algorithms from properties files (http://en.wikipedia.org/wiki/.properties). We have decided to provide this functionality from FADSE as well. A configuration example for NSGA-II is shown below:

crossoverOperator_ = SinglePointCrossover
mutationOperator_ = BitFlipMutation
selectionOperator_ = BinaryTournament2
populationSize_ = 100
maxEvaluations_ = 25000
mutationProbability_ = 0.16
crossoverProbability_ = 0.9

This has to be written in a properties file that is specified in the XML input file (see 3.6.4). All the algorithms have different structures of the properties files. We have provided examples in FADSE for: AbBYS, Densea, FastPGA, IBEA, NSGA-II, OMOPSO, PESA2, SMPSO, SPEA2.

3.9 Summary

In this chapter the basic functions of FADSE were introduced. The more advanced ones will be presented in Chapter 4. FADSE is a design space exploration framework that can be obtained for free from the project site (http://code.google.com/p/fadse/). If

is developed on top of the jMetal library, which includes many heuristic search algorithms.

First we have made an analysis of the state of the art DSE frameworks publicly available and concluded that they miss some important features.

FADSE comes to solve problems existent in other tools:

- few DSE algorithms;
- impossible to choose/configure the DSE algorithm;
- hard to configure;
- lack of distributed evaluation to accelerate DSE. Also lack of the database integration which leads to a great time reduction for the DSE process;
- bounded to certain simulators;
- metrics to evaluate the results are not integrated.

To solve the first problem we have integrated FADSE with the jMetal library. This is an advantage because FADSE can benefit from the experience of other researchers in the domain of heuristic algorithms. Since jMetal is actively developed, any new algorithms (bug fixes) that will be included in this library will be automatically available in FADSE. The included algorithms can easily be configured from external files. In fact, the entire framework can be easily configured through an easy to use, human readable XML file.

The XML interface allows the user to specify the simulator he/she wants to use for the DSE process, the parameters that he/she wants to vary and of course the objectives that need to be optimized.

The user can take advantage of the available resources and distribute the evaluation process in different computers connected in a network. Using VPNs, the network can spread to many computers located at great distances, the only requirement being an internet connection. The traffic generated by FADSE is very small so high speed connections are not necessary.

Through the integration with a database, FADSE can decrease substantially the required time for a DSE process. Simulations from previous explorations or even the current exploration can be reused, avoiding simulating again the same configuration.

The flexible XML interface, alongside with the connector model of integrating simulators, allows any existing simulator to be integrated into FADSE. This makes FADSE a generally available tool for design space exploration.

FADSE conveniently integrates metrics that are compatible with its outputs to make the user's job of analyzing the quality of results and the quality of the DSE process easy. The results are presented in a common format understood by all the office suites on the market.

More details on how to use, configure or develop FADSE can be found in our technical report "Developing a framework for ADSE which connects to multicore simulators" [55].

"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 Related Work

In this chapter we present different methods to include knowledge into the DSE algorithms, to help them find better results faster. One of the simplest methods is through constraints. The work we have done in this field is inspired from the MultiCube (M3Explorer) project. We have extended their work and implemented new features to make the interface more powerful and flexible.

We also needed to implement hierarchical parameters (gene, sub-gene) inside FADSE to allow the user to specify relations between them. The authors of ArchExplorer [47] present a similar feature in their DSE tool, but no details are provided about the actual implementation.

Our final approach was to use human generated fuzzy rules that will be interpreted by FADSE and given to the DSE algorithm. To our knowledge we are the first to use fuzzy as a method to allow the human expert to express knowledge about computer architectures in a comprehensible format that will help the DSE process perform better. There are various methods to obtain/use knowledge to accelerate the DSE process. In [56] fuzzy logic is used, during the design space exploration process, to provide an estimation of the objectives before simulating the individuals. In this approach, the evolutionary algorithm progresses normally, during this time a fuzzy system learns from the simulated individuals. When the confidence in the system is high, the individuals are first sent to the fuzzy system. If the objectives estimated are good enough, only then the individuals are sent for simulation. The confidence is considered sufficient either after a fixed number of generations or after the error of the estimation is low enough.

A similar approach to the one above is undertaken in [57] where neural networks are used to learn the data and to predict the values of the objectives.

4.2 Design Space Constraints

4.2.1 Motivation

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. The idea was to make FADSE and M3Explorer compatible at the interface level. We wanted to make the migration of any simulator compatible with M3Explorer to FADSE very easy. Since then, FADSE has evolved and has many more features, but migrating from M3Explorer to FADSE should not be difficult since FADSE's interface is mostly a superset of the M3Explorer interface.

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).

4.2.2 Implementation in FADSE

In FADSE, constraints are expressed in the XML input interface as "rules". There are three types of rules that can be specified in the XML input. We have implemented these rules as it is shown below.

Relational rule:

The most common type of rule is used to express relations between parameters or parameters and constants. An example is shown below where we want the level 2 cache size to be larger than 2048KB:

```
<rule name="minimum cache size">
        <greater-equal>
                <parameter name="l2_cache_size"/>
                <constant value="2048"/>
        </greater-equal>
 </rule>
```

In the relation above we can use: <*greater*>, <*greater-equal*>, <*less*>, <*less-equal*>, <*equal*>, <*not-equal*>. The parameter name must match a parameter that was defined in the XML in the parameters tag. The second argument of the rule can be a parameter or a constant value.

We have extended this further through the integration of expressions. Now the user can specify mathematical operations inside the rules:

```
<rule name="l2 larger than l1">
        <greater>
                <parameter name="dl2_nsets*dl2_bsize*dl2_assoc"/>
                <parameter name="  dl1_nsets*dl1_bsize*dl1_assoc+
                                   il1_nsets*il1_bsize*il1_assoc"/>
        </greater>
</rule>
```

In the example above we want to have the level 2 cache size bigger than level 1 data cache size and level 1 instruction size combined. Without the use of

expressions, the cache size could not have been expressed (since it is a combination of multiple parameters) and also the sum between level 1 caches could not have been performed.
This is an extremely powerful feature. The user can use functions like: * +, -, *, /, ^ (power: 2^3 = 8), % (modulo), cos, sin, tan, acos, asin, atan, sqrt (the square root), sqr (the argument multiplied by itself), log (logarithm), min, max, ceil (upper rounding of the number), floor (truncate), abs (absolute value), neg (negative), rndr (random).

Constant values can be used inside expressions. The following construct becomes valid:

```
<parameter name="2048"/>
```

Thus the *constant* tag can be replaced with constant expressions. The effect will be the same.

If rule:

Sometimes certain constraints should be imposed only in certain situations. For this we implemented the *If* rule:

```
<rule name="cache associativity limitation">
    <if>
        <greater-equal>
            <parameter name=" l2_cache_assoc "/>
            <constant value="1"/>
        </greater-equal>
        <then>
            <equal>
                <parameter name=" l1_cache_assoc "/>
                <constant value="1"/>
            </equal>
        </then>
    </if>
</rule>
```

In the example above we want the level 1 cache to have an associativity of 1 if the level 2 cache has an associativity of 1.

And/or/xor rule:

There are situations when more complex conditions have to be built inside an *If* rule. The *And/or/xor* rules can be used to achieve this:

```
<rule name="cache size check">
    <and>
        <greater-equal>
            <parameter name="l2_cache_size"/>
            <parameter name="l1_cache_size"/>
        </greater-equal>
        <greater-equal>
            <parameter name="l2_cache_size"/>
            <constant value="512"/>
```

```
            </greater-equal>
        </and>
    </rule>
```

In the above example we want the level 2 cache size to be bigger than the level 1 cache size and also level 2 cache size to be bigger than 512kB. The "and" tag can be replaced with "or" or "xor".

Any number and any type of rules (*Relational* rules, *If* rules and also *And/or/xor* rules) can be inserted in an *And/or/xor* rule.

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

In Paragraph 2.5 we presented the constraints and the approach taken into NSGA-II to solve it. In the proposed implementation the number of constraints that an individual broke influenced the selection process. We have the same approach but an *And/or/xor* and an If rule counts as one, even if there are multiple embedded *Relational* rules. For the future we plan to compute a distance from the border, normalize the distances to all the borders and add them up. This way a better selection process might be performed.

4.3 Hierarchical Parameters

4.3.1 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.

Before moving forward some notations have to be established:

- Parameters are depicted using the following format: *p#* where *#* is replaced by a number (an id that uniquely identifies the parameter), e.g. *p2*.
- Invalidation values are depicted as: *i#* where *#* is a number that specifies for what value the parameter is invalid. If the value of a parameter is equal with the value specified by the *#* in the *i#* then the parameter linked to the current one becomes invalid, e.g. *i3* means that the child parameter will be invalidated if the parent has a value equal with *3*.
- The current values of the parameters are written as simple numbers. The parameters can have any integer value and usually in the examples they do not have a special meaning.

Several types of situations have been identified. There are parameters which validate or invalidate a single set of parameters (two value selectors). Such an example can be: perform function inlining (true/false). If this parameter is set to true then all the parameters of function inlining become valid and need to be considered; otherwise they should be discarded. Other types of parameters complicate the situation even further. A parameter which selects from multiple sets of parameters, invalidating the others can be also encountered (multiple value selectors). For this

situation the example of branch predictor type can be used. If there are 3 types of branch predictors each one of them with their own set of parameters then selecting one type must validate the corresponding set while deactivating the others. An instance of this situation might look like this: if branch predictor is of type 1 then we need to activate parameter *p1*, *p2* and *p3* and deactivate parameters *p4*, *p5*, *p6*, *p7* and *p8* (which correspond to type 2 and 3). If the branch predictor is of type 2 then only parameters *p4*, *p5* and *p6* must be valid while *p1*, *p2*, *p3*, *p7* and *p8* are invalid and so on.

It can be easily observed that the first situation (two value selectors) is a subset of the second one. So only the second one will be discussed.

Multiple levels of invalidation can exist. For example parameter *p1* validates or invalidates parameters *p2*, *p3* and *p4* and also parameter *p2* invalidates/validates parameters *p5* and *p6*. The resulting structure is a tree.

4.3.2 Proposed Parameter Representation

A straightforward solution to hierarchical parameters is to not take them into consideration. Since the parameters are invalid in some situations, the evaluate function will simply ignore them and return the values for the objectives. This approach will lead to some problems. The algorithm will generate (through mutation and crossover) many individuals which are considered to be different but in fact pointing to the same individual, thus polluting the population. In Figure 4.3-1 such a situation is presented: the parameters *p4*, *p5*, *p6* and *p7* are dependant on other parameters and they are all invalid for the first individual (gray border), but valid for the second one (black border) .When the crossover operator is applied, there is chance that the crossover point is somewhere within the invalid parameters. The first individual takes the parameters from the second one (*p6* and *p7* in Figure 4.3-1) and they become invalid because the parameters which decide their validity remain the same. So the first individual becomes a new individual from the perspective of the parameters (different genotype) but in fact it will have the same objective values as its predecessor (same phenotype). In a worse case, both individuals have invalid parameters and the crossover is performed between them. Then the algorithm obtains two individuals which are equal with the previous ones.

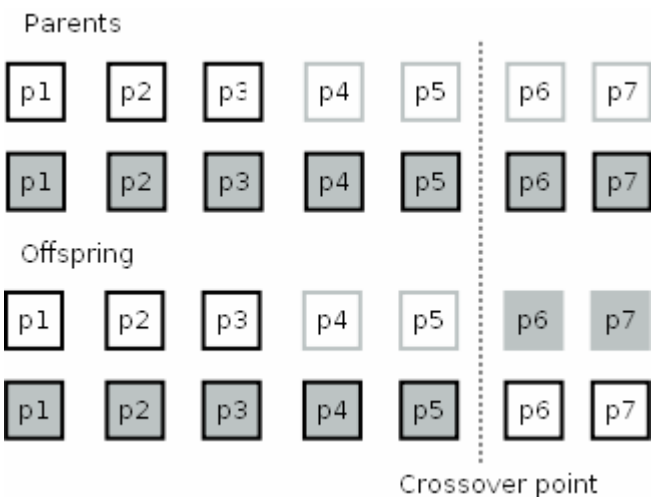


Figure 4.3-1 Problems that might arise when performing crossover on invalid parameters

If the crossover will produce the same individuals over and over again there is a chance that these individuals are very good and through elitism they will have greater chance to be selected in the next generation. They will produce new individuals, which are in fact identical, and the population will get filled with the same individual.

The mutation operator manifests the same problem. If the mutation operator picks an invalid parameter to mutate, it will create the same individual again.

To avoid these problems new operators are needed, which take into consideration the information of validity/ invalidity.

The proposed solution is to build a tree from the hierarchical parameters. When crossover is called, it will have available two trees, one for each individual. Using the superposition principle, the operator can determine the valid crossover points. The mutation operator will receive only one tree associated with the individual and it will choose a valid mutation point.

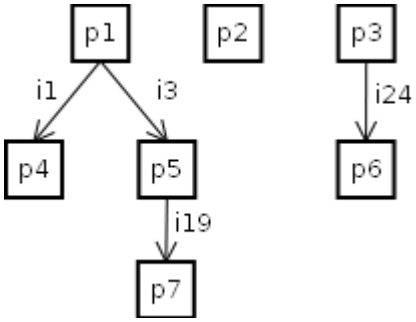


Figure 4.3-2 Tree representation with invalidation values

A tree that could have caused the situation presented in the chapter above could be the one in Figure 4.3-2. In this tree parameters *p4* and *p5* depend on parameter *p1*, parameter *p6* depends on parameter *p3* and parameter *p7* depends on parameter *p5*. Parameters *p1*, *p2* and *p3* are roots to different trees.

A parameter becomes valid or invalid because of the values of its parent. For example parameter *p4* becomes invalid when parameter *p1* has a value of 1 (*i1*). Parameter *p5* becomes invalid when parameter *p1* has a value of 3 (*i3*). This information must also be stored in the tree.

Using this representation will mean that parameter *p4* and *p5* can not be invalid at the same time (because parameter *p1* can be either 1 or 3). If for example parameter *p1* had another child (*p8*) there is no possibility to simulate a situation when only one parameter is valid at the same time (see Figure 4.3-3).

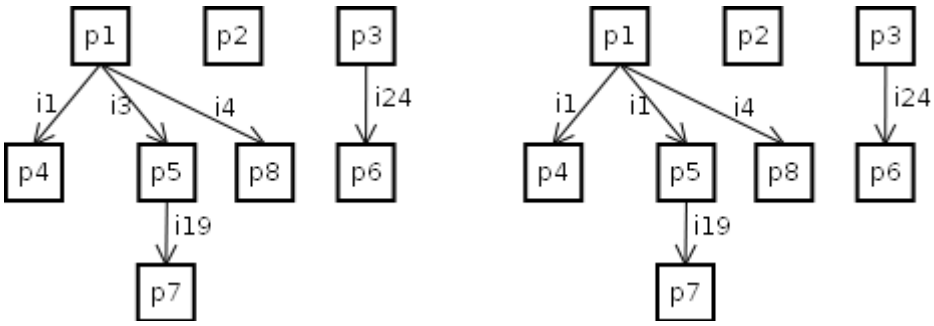


Figure 4.3-3 If only one value is possible for invalidation, some real life situations might be impossible to represent

Parameter *p1* can have only one value at the same time, so making parameter *p8* the only child valid is possible by setting the invalidity values for *p4* and *p5* to *i1*. But then parameter *p5* can not be the only one valid at a certain time. This situation can be encountered for example when parameter one chooses the type of branch predictor (as explained in the chapters above). To allow these situations, multiple values have to be accepted on each edge of the tree. So parameter *p4* will be invalid when parameter *p1* is either 3 or 4, parameter *p5* will be invalid when parameter *p1* has a value of 1 or 4 and parameter *p8* will be invalid when parameter *p1* is 1 or 3.

For simplicity only a tree with a single value will be discussed in this chapter. Having multiple values does not change the implementation, only the way the tree is built and searched.

4.3.3 Adapting Genetic Operators

4.3.3.1 Crossover

The crossover operator receives the tree and two individuals and switches from the current encoding of the individual to a tree encoding. At the same time it determines the valid and invalid edges (see Figure 4.3-4).

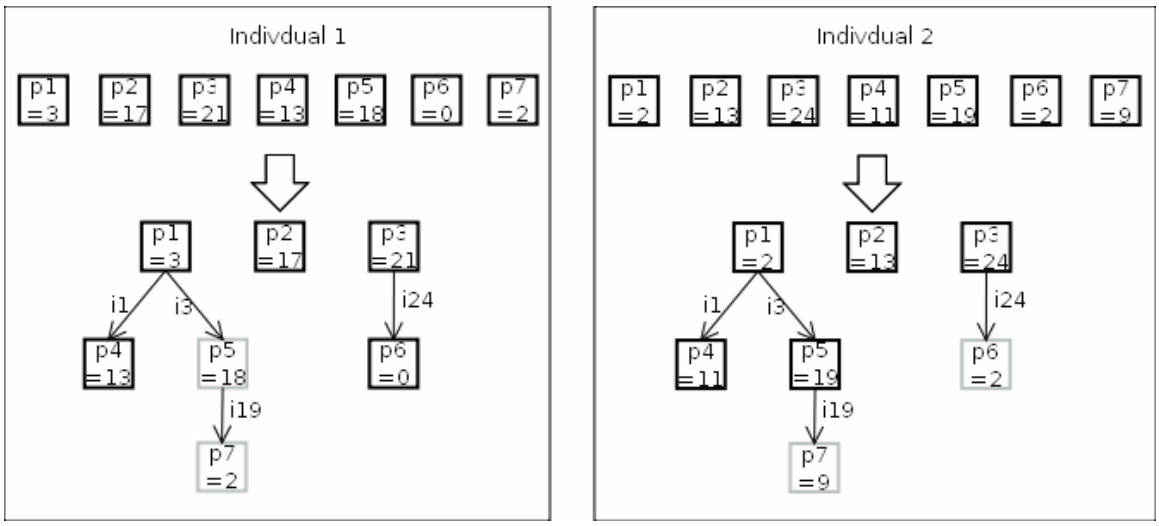


Figure 4.3-4 Converting an individual to a tree representation

Figure 4.3-4 shows two individuals being inserted in the tree (values under the boxes represent the current values of the parameters). The first individual has parameter *p1* set to value 3 so parameter *p5* with its entire children become invalid (gray borders), regardless of the value of parameter *p5*. For the second individual parameters *p7* and *p6* are invalidated.

The next step for the crossover operator is to determine the valid edges where crossover can be performed (single point crossover is assumed in this example). Crossover can be performed on edges that do not point to an invalid node or on the root nodes. If this would not be respected and the edge, above parameter p5, would be selected then parameter p5 and p7 from the second individual will be moved to the first one and they will become invalid. This way, the first individual remains the same after crossover. Programmatically this can be done like this:

- obtain an two arrays of binary values from the trees that specify if the node is valid or not;
- perform an AND operation between the binary arrays;
- randomly select a point from the array that has the value set to true.

For the example above we would have the following arrays: for individual 1 - (1 1 1 1 0 1 0), for individual 2 – (1 1 1 1 1 0 0). After the AND operation the array obtained will be: (1 1 1 1 0 0 0). This means that the crossover can be done using the roots (first three values of 1) and also on node p4 (the forth value of 1 in the array). Select a value between 1 and 4 and perform the tree crossover using that value.

Since the representation is not a single tree, to simplify the implementation, a virtual root node should be considered, which has as children all the roots in the

original representation. Using this representation means that, when a value is selected as the crossover point, the edge, above the chosen node, will be used (see Figure 4.3-5 where parameter p1 is selected as a crossover point).

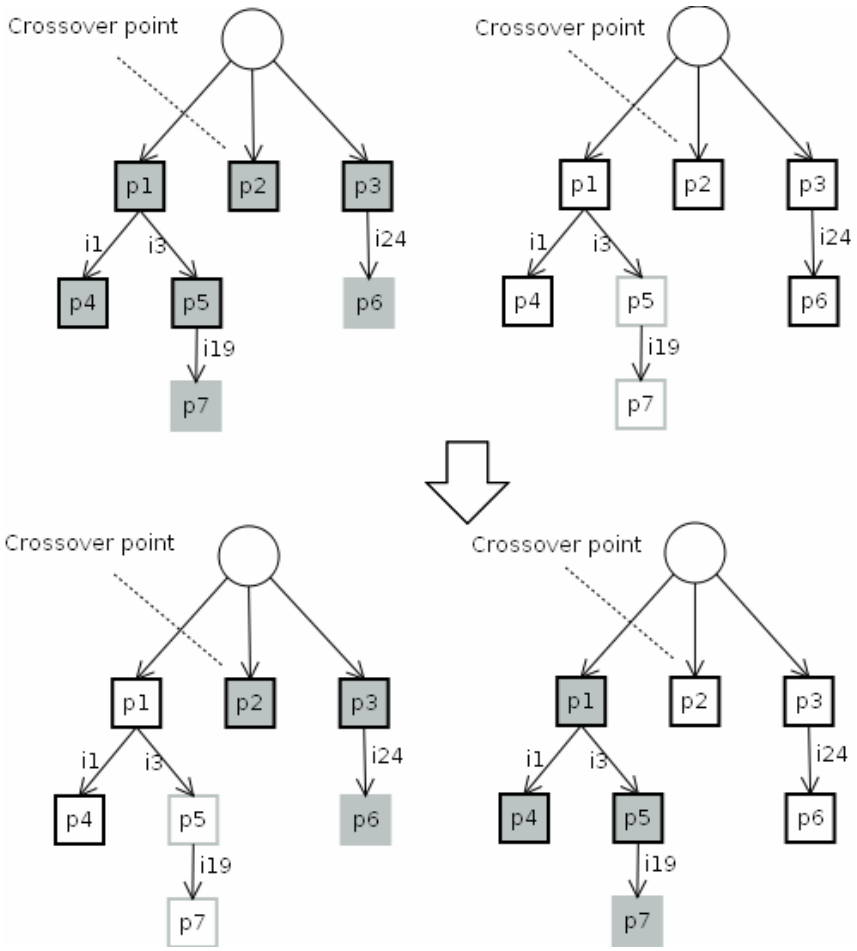


Figure 4.3-5 Tree crossover

If an edge has multiple values associated to it, the algorithm will have to search through all of the values to determine if the child value is valid or not.

4.3.3.2 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.

4.3.4 Implementation in FADSE

To support hierarchical parameters, in FADSE, a new crossover and mutation operator had to be implemented. A data structure to hold the tree was also necessary and additions to the XML interface had to be made so that the user can describe these trees in an easy manner.

In this paragraph the focus will be on the XML interface. We will present the <*relations*> tag (see 3.6). This tag can contain any number of relations. A relation is described within a <*relation*> *(*note the missing "s"*)* tag like the one below:

```
<relation>
        <if parameter="…" value="…">
                <then_invalidate>
                        <parameter name="…"/>
                        …
                </then_invalidate>
        </if>
</relation>
```

In the *if* tag attributes, the name of the parent parameter and the value that affects the child/children are specified. In the *then_invalidate* tag, a list of parameters is given which is invalidated when the parent has the specified value.

The names of the parameters must coincide with the names provided in the <*parameters*> tag. If a parameter deactivates, for the same value, multiple parameters then these parameters can be specified as a list in the same <*then_invalidate*> tag.

If a parameter (or parameters) is invalidated by multiple values of the same parent parameter then two relations have to be constructed, like the example below:

```
<relation>
        <if parameter="p1" value="1">
                <then_invalidate>
                        <parameter name="p4"/>
                        <parameter name="p8"/>
                </then_invalidate>
        </if>
</relation>
<relation>
        <if parameter="p1" value="2">
                <then_invalidate>
                        <parameter name="p4"/>
                        <parameter name="p8"/>
                        <parameter name="p9"/>
                </then_invalidate>
        </if>
</relation>
```

In this situation, parameters p4 and p8 will be invalid if parameter p1 is either 1 or 2 but parameter p9 will be invalid only when parameter p1 has a value of 2.

An invalid parameter can **not** have multiple parents.

If lists of strings are used in the parameters list and they determine if another parameter is valid or not, then the position of the string (**starting with 0**) should be used as a selector.

To use the hierarchical parameters besides defining them in the XML file, the user has to specify the correct mutation and crossover operator in the algorithm configuration file (properties file).

4.4 Introducing Domain-specific Knowledge through Fuzzy Logic

An evolutionary algorithm designed for a specific problem usually performs better than a general one. Its drawback is that it can be used only for that particular problem. Because of this, specific algorithms can not be included into FADSE since its aim is to be a general DSE framework. Our idea is to let the human expert influence the algorithm before running an exploration. The problem is how could an expert express his/hers knowledge into a format understandable by both designer and an evolutionary algorithm. In previous paragraphs we have presented some methods: constraints and hierarchical parameters. Those were very deterministic, but very often the designer has intuitions about the relations between parameters. He/she knows that if a parameter is big, another one should be small and so on. They work with linguistic terms. We are using fuzzy logic to express this knowledge. This paragraph provides a short introduction to fuzzy logic and then it presents how it was integrated into FADSE.

4.4.1 Fuzzy Logic Overview

Fuzzy logic extends the classical logic by allowing intermediate values of membership function (between 0 and 1) to be used. Fuzzy logic enables the computers to work with vague linguistic terms and statements such as: if the number of transistors available is *large* then integrate *many* cores in the processor.
Fuzzy logic is based on the theory of fuzzy sets introduced by Lotfi A. Zadeh [58].

4.4.1.1 Fuzzy Sets

Georg Cantor and Richard Dedekind are considered the parents of modern set theory. A set is a collection of objects. Objects can be anything, from numbers to cars and books. The objects from the set have to be definite and distinguishable. Definite means that if there is an object and a set, it must be possible to determine if the object is a member or not of the set. Distinguishable refers to the fact that it can be established if two objects are different or the same (given a certain set).

In classic set theory, an object is member of a set or it is not. So the proposition "x (an object) is member of the set X" is true or the negation of this proposition is true. This is called the law of excluded middle. In fuzzy sets a membership grade is assigned to each object.

Given a collection of objects X, a fuzzy set S in X is defined as a set of ordered pairs:

$$S \equiv \left\{ \langle x, \mu_S(x) \rangle \middle| x \in X \right\}$$

where $\mu_S(x)$ is called the membership function for the set of all objects x in X ($\equiv$ is read as "defined as" $\equiv$). $\mu_S(x)$ associates a real value in the interval [0,1] to each x. The notation for an ordered set is $\langle x, y \rangle$, while for an unordered set is $\{x, y\}$. So the classical set theory (called by the fuzzy set researchers as the crisp set theory) is a special case of fuzzy set theory where the membership values are restricted to 0 and 1.

The ordered pair $\langle x, \mu_S(x) \rangle$ is called a fuzzy singleton.

All the objects that can be taken into consideration when considering a set S is the universe of discourse (often called in short: the universe).

A membership function can be continuous or discrete. Continuous functions can have different forms. Common functions are: trapezoidal functions, triangular functions, Gaussian functions (see Figure 4.4-1).

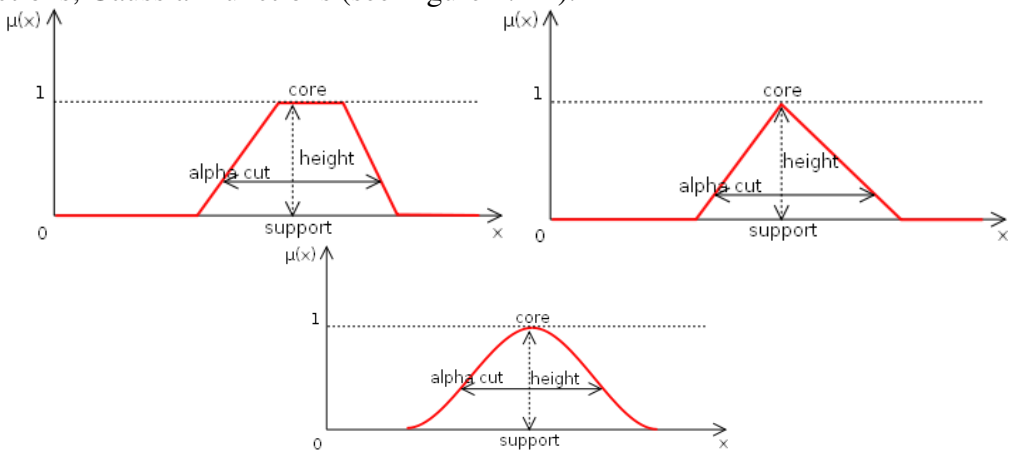


Figure 4.4-1 Common types of continuous membership functions

When talking about membership functions some basic notions have to be defined (see Figure 4.4-1):

- Support: elements having a non-zero degree of membership;
- Core: elements having a degree of membership equal with one;
- Alpha cut: set of elements having a degree of membership larger or equal with alpha;
- Height: maximum membership value.

Discrete fuzzy sets are defined using ordered pairs:

$$S = \{\langle x_1, \mu_S(x_1)\rangle, \langle x_2, \mu_S(x_2)\rangle, ..., \langle x_n, \mu_S(x_n)\rangle | x_i \in X, i = 1,2,...,n\}$$

where x_i are discrete variables and $\mu_S(x_i)$ is the membership value of the point x_i in the universe X.

Two fuzzy sets are equal if their associated membership functions are equal for all the objects in the set. A fuzzy set *A* is a subset (or is included) of another fuzzy set *B* if its membership function is smaller or equal than the membership function of *B* for all the objects in *A*.

4.4.1.1.1 Linguistic Variables and Values

A meaning can be associated to a fuzzy set. The label assigned to the domain (universe) of the fuzzy set is called **linguistic variable** (e.g. number of transistors, cache size, etc.). The labeled collection of fuzzy sets in this domain are called **linguistic values** (e.g few, many, small, medium, large – see Figure 4.4-2). Linguistic values are strongly dependent of the context (problem).

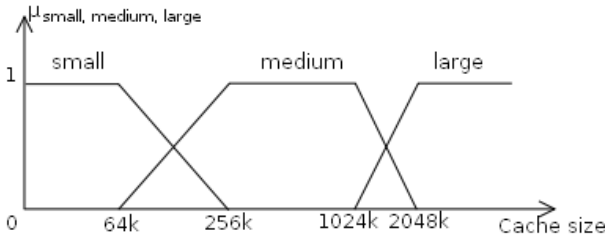


Figure 4.4-2 Linguistic terms. (The Cache size axis is not represented at scale)

Using linguistic variables allows working at different granular levels. The terms "small", "medium" and "large" are fuzzy granules. The specific values are not used anymore. Instead their linguistic value is used, the values being grouped by

similarity in these granules. This allows "computing with words". The granules can be obtained by using an expert in the domain or automatically. In most of our work we use experts' opinions due to the nature of our problem (see Chapter 6.3), but we also developed a method to obtain them automatically through some data mining methods (see Chapter 5.6).

4.4.1.1.2 Fuzzy Sets Operations

The elementary sets operations will be defined in this section: union, intersection, complement (see Figure 4.4-3) and Cartesian product.

Let A and B be two fuzzy sets defined on the same universe X. The union of these two fuzzy sets is defined as:

$$A \cup B \equiv \left\{ \left\langle x, \mu_{A \cup B}(x) \right\rangle \middle| x \in X \text{ and } \mu_{A \cup B}(x) = \max\left(\mu_A(x), \mu_B(x)\right) \right\}$$

The intersection of A and B is defined as:

$$A \cap B \equiv \left\{ \left\langle x, \mu_{A \cap B}(x) \right\rangle \middle| x \in X \text{ and } \mu_{A \cap B}(x) = \min\left(\mu_A(x), \mu_B(x)\right) \right\}$$

The fuzzy complement of A is defined as:

$\overline{A} \equiv \left\{ \left\langle x, \mu_{\overline{A}}(x) \right\rangle \middle| x \in X \text{ and } \mu_{\overline{A}}(x) = 1 - \mu_A(x) \right\}$; a generalization of the crisp (classical logic) complement.

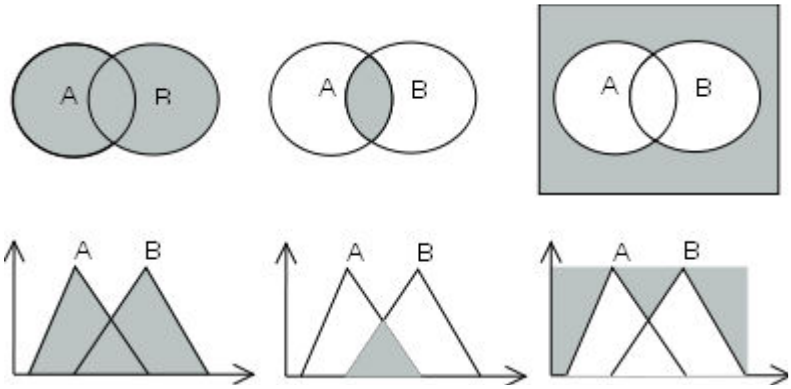


Figure 4.4-3 Comparison between set operations and their fuzzy equivalents [59]. The set operations are represented using Venn diagrams. The represented operations are: union, intersection and complement

Some of the rules belonging to classic sets still hold for fuzzy sets:

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$
$$\overline{A \cup B} = \overline{A} \cap \overline{B} \quad \text{(De Morgan's rules)}$$

These relations are true because the membership functions of the left and right hand side of the relations are equal. This is based on: $\mu_A(x) = \mu_B(x)$ if and only if A=B.

All the operations from the classical set theory, except the excluded middle axiom ($P \vee \overline{P}$ is true), are still valid for fuzzy sets (comutativity, associativity, distributivity, idempotency, identity, transitivity, involution):

For fuzzy sets the law of excluded middle can be extended like this:

$$A \cup \overline{A} \neq X$$
$$A \cap \overline{A} \neq \emptyset$$

4.4.1.1.3 Fuzzy Sets Binary Relations

If A and B are two fuzzy sets defined on X and Y respectively then the Cartesian product $A \times B$ of these two sets in $X \times Y$ is:

$$A \times B = \left\{ \left\langle \langle x, y \rangle, \mu_{A \times B}(x, y) \right\rangle \middle| x \in X, y \in Y, \mu_{A \times B}(x, y) = \min\left(\mu_A(x), \mu_B(y)\right) \right\}$$

Any other relation is a subset of the Cartesian product.

The composition of two binary relations R and S defined on $X \times Y$ and $X \times Z$ respectively is defined as:

$$R \circ S = \left\{ \left\langle \langle x, z \rangle, \bigcup_y \mu_R(x, y) \cap \mu_S(x, y) \right\rangle \middle| x \in X, y \in Y, z \in Z \right\}$$

The composition is usually called a max-min composition and it is the most used in the literature [60]. Multiplication is sometimes used instead of min and then it is called max-star composition.

4.4.1.2 Fuzzy Logic

In fuzzy logic a proposition can be true (1) or false (0) like in classic logic, but it can all be all the intermediate truth values between 0 and 1. Fuzzy logic tries to formalize the way humans are able to perform imprecise reasoning. In fuzzy logic as in classical logic the connectives "and" (conjunction $\wedge$), "or" (disjunction $\vee$), "if then" (implication $\Rightarrow$), "if and only if" (equivalence $\Leftrightarrow$) and the "negation" (complement $\overline{\ }$) have to be defined. Fuzzy logic theory is derived from the fuzzy set theory.

For the conjunction, disjunction and complement the definitions provided by Lotfi Zadeh in [58] can be used:

$$\mu_{A \wedge B}(x) = \min(\mu_A(x), \mu_B(x))$$
$$\mu_{A \vee B}(x) = \max(\mu_A(x), \mu_B(x))$$
$$\mu_{\overline{A}}(x) = 1 - \mu_A(x)\text{; this relation is identical with the one in classical logic}$$

Other definitions are possible. For example product for conjunction and bounded sum for disjunction:

$$\mu_{A \wedge B}(x) = \mu_A(x) \cdot \mu_B(x)$$
$$\mu_{A \vee B}(x) = \min\left(\mu_A(x) + \mu_B(x), 1\right)$$

Or:

$\mu_{A\wedge B}(x) = \mu_A(x) \cdot \mu_B(x)$

$\mu_{A\vee B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)$; These two relations are identical with the relations from classical logic.

The conjunction and disjunction are called norms. When min is used for conjunction and max for disjunction they are called the min-max norms. The conjunction is called a t-norm and the disjunction an s-norm or t-conorm. The negation remains the same in both cases.

Some conditions must be respected:

- for an AND - if the element has a membership of 1 on both sets, then it must have a membership of 1 on their intersection. If it has a membership of 0 in one set, the membership of the intersection of these two sets should be also 0.
- for an OR - if in both sets the membership of the object is 0, the membership of the object in the union must be 0 and if one is 1 then the membership in the union of the object must be 1.

In classical logic the implication is defined like this:

$A \to B = \overline{A \cap \overline{B}} = \overline{A} \cup B$

$A \to B = \overline{A} \vee B = \max\left(\overline{A}, B\right)$ (if the truth values are considered to be 0 and 1)

If the implication involves two universes then it is represented through a relation R. If A is a set which describes a proposition P defined on the universe X and there is another proposition Q described by set B defined on the universe Y then the implication $P \to Q$ can be defined as:

$R = (A \times B) \cup \left(\overline{A} \times Y\right) \equiv$ IF A, THEN B

IF $x \in A$ where $x \in X$ and $A \subset X$ THEN $y \in B$ where $y \in Y$ and $B \subset Y$

Extending this idea to fuzzy logic, Zadeh [61] proposed the following definition for implication in fuzzy logic (for min-max norms):

$A \to B = \overline{A} \vee B = \max\left(\overline{A}, B\right)$

where A and B are fuzzy sets defined on the same universe U.

If A is defined on universe X and B is a fuzzy set defined on the universe Y then the implication is a fuzzy set in $X \times Y$ with the membership function:

$\mu_{A\to B}(x, y) = \max\left[\left(\mu_A(x) \wedge \mu_B(y)\right), \left(1 - \mu_A(x)\right)\right]$ (Zadeh implication)

This can be obtained from $R = (A \times B) \cup \left(\overline{A} \times Y\right)$

$$R = (A \times B) \cup \left(\overline{A} \times Y\right) = \max\left[(A \times B), \left(\overline{A} \times Y\right)\right] = \max\left[\min(\mu_A, \mu_B), \min(1 - \mu_A, \mu_Y)\right]$$
$$= \max\left[\min(\mu_A, \mu_B), \min(1 - \mu_A, 1)\right] = \max\left[\min(\mu_A, \mu_B), 1 - \mu_A\right] = \max\left[\mu_A \wedge \mu_B, 1 - \mu_A\right]$$

because the membership of an object on the whole universe is 1 ($\mu_Y = 1$).

Implication can be defined in different ways depending on what tautologies from the classical logic need to be preserved in the fuzzy logic. This is why there are different definitions. In [62] 72 different possible definitions are found.

One was defined by Jan Lukasiewicz [63]. Lukasiewicz defines the implication and negation and derives the other norms ($\wedge,\vee$):

$$\mu_{A\to B}(x,y) = \min(1,1-\mu_A(x)+\mu_B(y))$$

and the negation:

$$\mu_{\bar{A}}(x) = 1-\mu_A(x)$$

Then using $A \vee B = \bar{A} \to B$

$$\mu_{A\vee B}(x) = \min(1,\mu_A(x)+\mu_B(x))$$

Using $A \wedge B = \overline{\bar{A} \vee \bar{B}}$

$$\mu_{A\wedge B}(x) = 1-\min(1,1-\mu_A(x)+1-\mu_B(x)) = \max(0,\mu_A(x)+\mu_B(x)-1)$$

One of the most used implications in fuzzy logic when the sets are defined on different universes is the one proposed in [64], which is also defined for min-max norms:

$$\mu_{A\to B}(x,y) = \min(\mu_A(x),\mu_B(y))$$ (Mamdani implication)

The Mamdani implication is the one used by us in the FADSE tool.

4.4.1.2.1 Generalized Modus Ponens

In fuzzy logic inference can be written as (hypothesis/conclusion):

$$\frac{x \text{ IS } A' \qquad \text{IF } x \text{ IS } A \text{ THEN } y \text{ IS } B}{y \text{ IS } B'}$$

Where A' is the membership function for x. A' does not have to be the same with the A in the antecedent of the implication. So this can be written as:

$$\frac{\mu_{A'}(x) \qquad A \to B}{\mu_{B'}(x)}$$

Depending on which implication is used there are different methods for calculating B': if the min-max norm is used then the inference is called a **joint-constraint** and B' is computed like this:

$$B' = A' \wedge (A \times B)$$

$$\mu_{B'} = \sup_u(\min(\mu_{A'}(u),\mu_{A\times B}(u,v)))$$

If A' is very similar with A then B' will be very similar with B. If A' does not resemble A then $\mu_{B'} = 0$ (B' is an empty set).

If the Lukasiewic implication is used then it is called a **conditional constraint** and B' is computed like this:

$$\mu_{B'} = \sup_u(\min(\mu_{A'}(u), 1 - \mu_A(u) + \mu_B(u)))$$

If A' is very similar with A then B' will be very similar with B. If A' does not resemble A then B' will be generic: $\mu_{B'} = 1$.

4.4.1.2.2 Rules

There are two types of fuzzy rules:

1) Qualified rules that assign additional constraints to rules (such as "very likely" or "not very true"). These types of rules will not be addressed here;
2) Categorical rules, that will be presented below.

The categorical rules are divided also into two categories: Takagi-Sugeno [65] models and Mamdani rules [66] [1].

Takagi-Sugeno rules have a form like the one below:

IF x_1 IS A_1 AND … x_n IS A_n THEN y=f(x)

where A_i are linguistic values of the input vector x. In this situation to the output a crisp equation is assigned. The degree of membership of the function is in this situation:
$w = \min\{\mu_{A_1}(x_1), \mu_{A_2}(x_2), ..., \mu_{A_n}(x_n)\}$ (an AND operation is performed). The output will be a fuzzy singleton $\langle y, w \rangle$.

If several rules exist then the fuzzy singletons are combined using a weighted sum:

$$y = \frac{\sum_{i=1}^{n} w_i \cdot y_i}{\sum_{i=1}^{n} w_i}$$

where n is the number of rules.

Mamdani rules have a form like the one below:

IF x_1 IS A_1 AND … x_n IS A_n THEN y IS B

where A_i and B are linguistic values of the input vector x and the output variable y, respectively. The part before "THEN" is called an antecedent and the part after a consequent. The difference from Takagi-Sugeno rules is that here the consequent is also fuzzy. Usually the min norm is used for "AND". The rule above can be written as:

[1] according to other sources there are three common methods of deductive inference: Mamdani systems, Sugeno models, and Tsukamoto models [67]

$$\mu_{A_1}(x_1) \wedge \mu_{A_2}(x_2) ... \wedge \mu_{An}(x_n) \rightarrow \mu_B(y)$$

If the min norm is used for AND then

$$\mu_{A_1}(x_1) \wedge \mu_{A_2}(x_2) ... \wedge \mu_{A_n}(x_n) = \min(\mu_{A_1}(x_1), \mu_{A_2}(x_2), ..., \mu_{An}(x_n))$$

In most of the situations A_i and B are not defined on the same universe. **This work uses the Mamdani implication** so the output membership function $\mu_{B'}$ can be obtained using:

$$\mu_{B'}(y) = \min(\mu_B(y), \mu_A)$$
where $\mu_A = \min(\mu_{A_1}(x_1), \mu_{A_2}(x_2), ..., \mu_{A_n}(x_n))$

It can be noticed that there are no OR and NOT operations inside the rule. If an OR is present the rule is broken into two rules, like shown below:

IF x_1 IS A_1 AND x_2 IS A_2 OR x_3 IS A_3 AND x_4 IS A_4 THEN y IS B

This can be re written as:

(IF x_1 IS A_1 AND x_2 IS A_2) OR (IF x_3 IS A_3 AND x_4 IS A_4) THEN y IS B

This rule can be broken into two rules:

IF x_1 IS A_1 AND x_2 IS A_2 THEN y IS B
IF x_3 IS A_3 AND x_4 IS A_4 THEN y IS B

In this situation if the max norm is used for OR the membership function can be calculated:

$$\mu_{B'}(y) = \max(\min(\mu_B(y), \min(\mu_{A_1}(x_1), \mu_{A_2}(x_2))), \min(\mu_B(y), \min(\mu_{A_3}(x_3), \mu_{A_4}(x_4))))$$

If a NOT is present in the rule:

IF x_1 IS NOT A_1 THEN y IS B

It can be written in two variants:

IF x_1 IS $\overline{A_1}$ THEN y IS B

or

IF $\overline{x_1}$ IS A_1 THEN y IS B

In conclusion any fuzzy implication can be written through a set of fuzzy rules which contain only the AND operator. This set is called **fuzzy logic rule base** and it can be written in general form like this:

R1: IF x_{11} IS A_{11} AND … x_{1n} IS A_{1n} THEN y_1 IS B_1

R2: IF x_{21} IS A_{21} AND … x_{2n} IS A_{2n} THEN y_2 IS B_2
…
Rm: IF x_{m1} IS A_{m1} AND … x_{mn} IS A_{mn} THEN y_m IS B_m

It must be noted that the rules are not required to have the same number of terms in the antecedent and some or all y_i can be the same linguistic variable.

A fuzzy logic rule base has to satisfy some conditions: it has to be complete and consistent [60].

A complete rule base means that no possible conditions are left out. The rule base below does not satisfy this condition:

IF x > 0 THEN y < 0
IF x = 0 THEN y = 0

because no rule is specified if x<0.

For a rule base to be consistent, the rules contained must not contradict each other. The example below is a system which is inconsistent (but complete):

IF x > 0 THEN y < 0
IF x > 0 THEN y = 0
IF x = 0 THEN y < 0
IF x < 0 THEN y > 0

4.4.1.2.3 Rule Aggregation

Each rule that has on the antecedent a membership value greater than zero will give an output in form of a membership function. These output functions need to be aggregated into a single output. The most used one is the Mamdani aggregation method [68]. In this method the output is equal with the maximum (MAX) truth value in the area were the output membership functions overlap.

4.4.1.2.4 Defuzzification

It is usually necessary that the output of the system to be a crisp value. Even if fuzzy sets are used, it is necessary to obtain a crisp output value to make a binary decision.

Defuzzification is the conversion of a fuzzy quantity to a precise quantity. There are many methods to do this: max membership value (the height method), center of gravity (center of area, centroid method), weighted average method, mean max membership and many others [69]. We have used only the center of gravity method. Center of gravity is one of the most used methods and is compute using the following equation:

$$z^* = \frac{\int x \cdot \mu(x) dx}{\int \mu(x) dx}$$

where $\mu(x)$ is the membership function.

4.4.1.2.5 Mamdani Rules-system

From all the possible inference systems we have focused on the Mamdani inference system [64]. For this we have used min and max for AND and OR respectively. For

the implication we used the Mamdani implication (MIN). For consequent aggregation we used the MAX function. Examples of how these are applied will be given in the following chapters.

4.4.2 Integrating Fuzzy Logic into FADSE

4.4.2.1 FCL Language

The Fuzzy Control Language (FCL) [70] will be used throughout this section to describe fuzzy functions. FCL is a standard language for fuzzy control programming and has been published by the International Electrotechnical Commission (IEC) [71]. The language specification can be found in IEC document 61131-7. A draft version can be found at [70]. We integrated the jFuzzyLogic library [72] 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.

Two input membership functions will be used as an example and one output function. The values used were chosen so that they would be easy to compute and to observe, they should not be taken as real possible values and are meant only as an example.

4.4.2.1.1 Input Variables

We defined the following input variables: level 1 data cache size (l1cache) and number of available transistors (transistors). The output variable is the size of the level 2 cache (l2cache). In the FCL language we can define these variables like this:

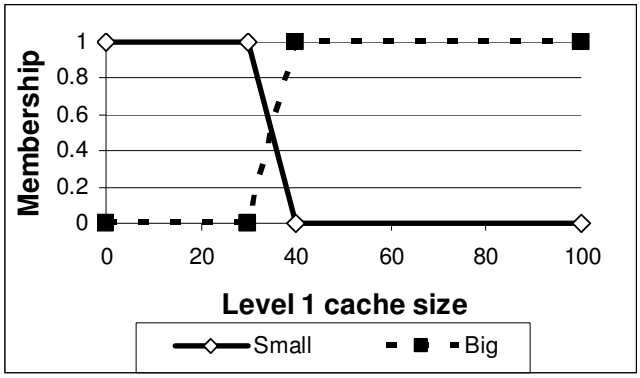


Figure 4.4-4 Graphical representation of the membership functions associated to level 1 cache size

```
VAR_INPUT
        l1cache : REAL;
        transistors : REAL;
END_VAR

VAR_OUTPUT
        l2cache : REAL;
END_VAR
```

The l1cache varies from 0 to 100 and it has two membership functions associated: one for small size and one for big size (linguistic values). The functions are shown in Figure 4.4-4.

These membership functions are described in FCL language like this:

```
FUZZIFY l1cache
        TERM small := (0, 1) (30, 1) (40, 0) ;
        TERM big := (30, 0) (40,1) (100,1);
END_FUZZIFY
```

The l1cache is considered to be small if it is under 30 (between 30 and 40 the membership function decreases) and big if it is over 40. This is a trapezoidal representation.

The same way we defined transistor count function in FCL format like this:

```
FUZZIFY transistors
        TERM few := (0, 1) (180, 1) (200,0) ;
        TERM many := (180,0) (200,1) (240,1);
END_FUZZIFY
```

4.4.2.1.2 Output Variables

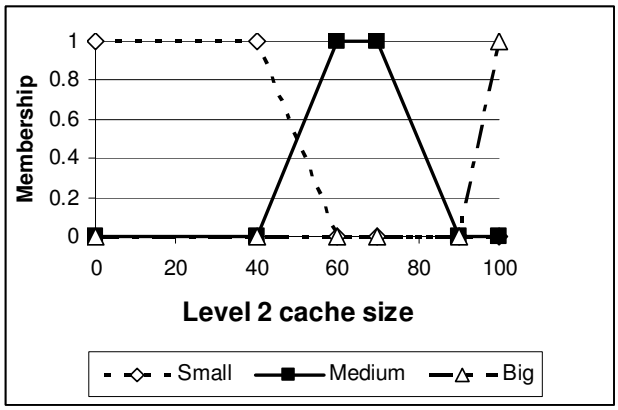


Figure 4.4-5 Graphical representation of the membership functions associated to output variable: level 2 cache size

We then assign some linguistic values to the l2cache. Level 2 cache size is split in three categories: small (0-40-60), medium (40-60-70-90) and big (70-100). The graphical representation can be seen in Figure 4.4-5.

The membership function associated to the level 2 cache can be described in FCL like this:

```
DEFUZZIFY l2cache
        TERM small := (0,1) (40,1) (60,0);
        TERM medium := (40,0) (60,1) (70,1) (90,0);
        TERM big := (70,0) (100,1);
        // Use 'Center Of Gravity' defuzzification method
        METHOD : COG;
        // Default value is 70- (if no rule activates the defuzzifier)
        DEFAULT := 70;
END_DEFUZZIFY
```

Along with the usual terms two other constructs are required: METHOD and DEFAULT. The METHOD specifies the defuzzification method (see 4.4.1.2.4). In this example the center of gravity is used (other possible methods are: right most max, center of area, left most max, mean max). The jFuzzyLogic library approximates the center of gravity using the following equation:

$$COG_{approx} = \frac{\sum_{j=0}^{\max\_x} x_j \cdot \mu_j(x_j)}{\sum_{j=0}^{\max\_x} \mu_j(x_j)}$$

where $\mu_j(x_j)$ is the membership value of point x_j.

The DEFAULT specifies a value that will be used if the rules system is incomplete and no rule is specified for the current interval.

4.4.2.1.3 Rules Representation

Te next step is to define the rules. The rules are:

- R1: if the l1 cache is small and the number of transistors big (many) then l2 cache can be big
- R2: if l1 cache is small but we do not have many transistors then the l2 should be medium is size
- R3: if l1 cache is big and we have many transistors at our disposal we can make the l2 cache a medium size
- R4: if the l1 cache is big and we do not have many transistors then the l2 cache should be small

  The rules above can be expressed in FCL like this:

```
RULEBLOCK No1
        AND : MIN;
        ACT : MIN;
        ACCU : MAX;
        RULE 1 : IF l1cache IS small AND transistors IS many
                THEN l2cache IS big;
        RULE 2 : IF l1cache IS small AND transistors IS few
                THEN l2cache IS medium;
        RULE 3 : IF l1cache IS big AND transistors IS many
                THEN l2cache IS medium;
        RULE 4 : IF l1cache IS big AND transistors IS few
                THEN l2cache IS small;
END_RULEBLOCK
```

In the rule block, the operators to use for AND and OR, the implication method and the aggregation method are defined. The AND/OR operators are defined in pairs so if AND is set to use the MIN function then OR will automatically use MAX.

The implication method is called activation in jFuzzyLogic (ACT) and the Mamdani implication is used in this example – MIN.

Aggregation is called accumulation in JFuzzyLogic and we have set it to use the MAX function. Many other norms, implications and aggregation methods are defined.

4.4.2.2 A simple example

To make it easier to understand we will represent the rules from Section 4.4.2.1.3 in a graphical form. On these rules we apply the inputs: l1cache = 32 and transistors = 185 (the values are for this example only).

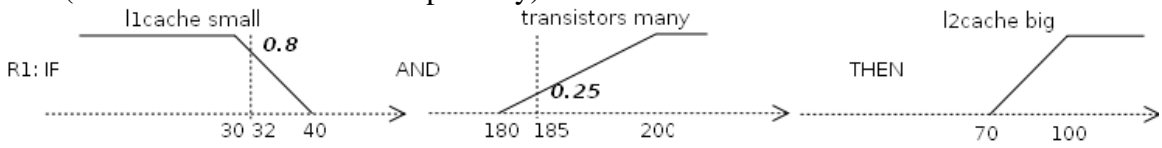


Figure 4.4-6 Rule 1 in graphical form with applied input

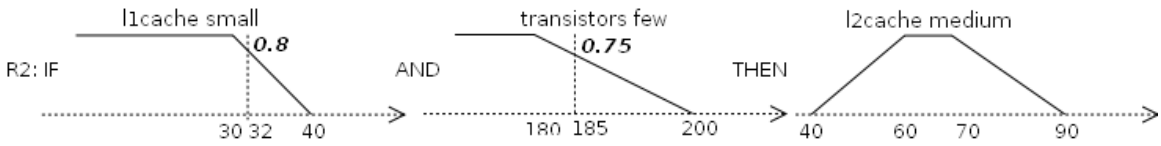


Figure 4.4-7 Rule 2 in graphical form with applied input

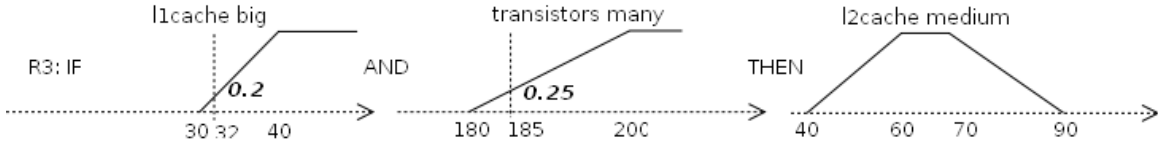


Figure 4.4-8 Rule 3 in graphical form with applied input

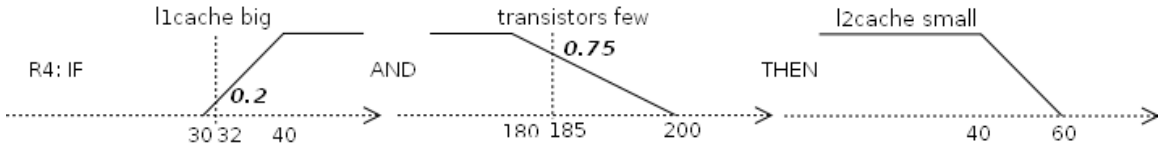


Figure 4.4-9 Rule 4 in graphical form with applied input

For each rule the intersection point is computed (see Figure 4.4-7, Figure 4.4-8, Figure 4.4-9 and Figure 4.4-9). For example the 32 value has a membership of 0.8 (80%) on l1cache small and 0.2 (20%) on l1cache big. The value 185 for transistors has 25% membership on the many membership function and 75% on the few.

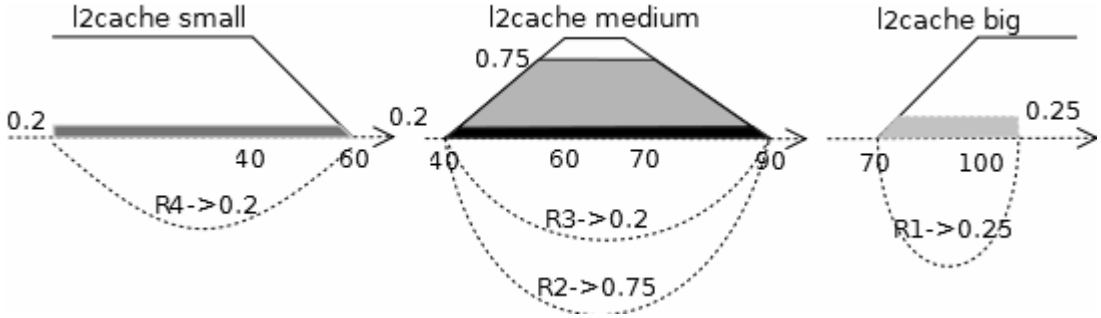


Figure 4.4-10 Output of the rules before aggregation

Since the MIN function is used for the AND operation, in rule 1 (R1) the value obtained will be 0.25 (min(0.8,0.25)). From R2, R3 and R4 we will get 0.75, 0.2 and 0.2 respectively. These values are used for implication.

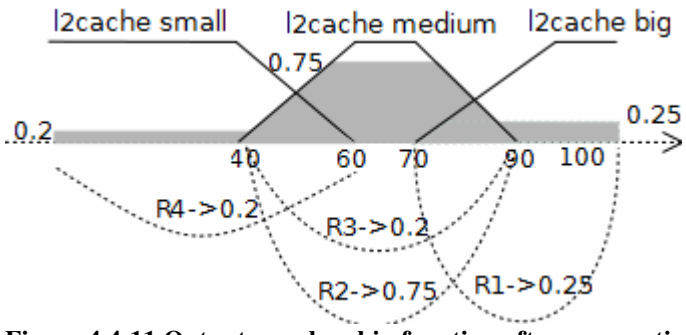


Figure 4.4-11 Output membership function after aggregation

Applying the activation (implication)

We are using min function for implication (Mamdani implication). This means that the output after implication will be the minimum between the value obtained after applying the AND operator and the output function. The result will look as in Figure 4.4-10.

Applying the accumulation method (rule aggregation)

We use the MAX for aggregation. We overlap the membership functions obtained in the previous step and only the maximum value remains.

The final form of the membership function is presented in Figure 4.4-11.

Defuzzification

On the obtained function we apply the approximated center of gravity as a defuzzification method. The value obtained for COG_{approx} is 57.56.

Since the jFuzzylogic library does not allow the same names for variables to be used for input and for output, a convention was made that to all the variables that are used as output variables an "out" prefix is concatenated. For example if *level1_cache_size* is used as an output parameter in one of the rules (it is used in the consequent) its name will be changed to *outlevel1_cache_size* in the FCL file (fuzzy control language file). The name of the parameter in the input XML remains the same (level1_cache_size in our example). In this way the same parameters can be used as inputs in some rules and as outputs in others:

IF slvp_size IS big THEN outlevel1_cache_size IS small
IF level1_cache_size IS medium THEN outl2_cache IS big

4.4.2.3 Mutation Operators

4.4.2.3.1 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 presented in Chapter 2.1.3.1 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.

4.4.2.3.2 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. Thus the pseudo-code becomes:

1. For all the variables in the individual;
   1.1. Generate a random number between 0 and 1;
   1.2. If the random number is smaller than the probability of mutation;
      1.2.1. If fuzzy rules exists for this parameter;
         1.2.1.1. Change the parameter to the value specified by the fuzzy rules;
      1.2.2. Otherwise (bit flip mutation);
         1.2.2.1. Change the current variable to a random value;
2. STOP.

4.4.2.3.3 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 (explained below). The bit flip mutation fuzzy looks like this:

1. For all the variables in the individual;
   1.1. If a fuzzy rule exists for this parameter;
      1.1.1. Compute COG of this variable taking into consideration the current values of the other variables;
      1.1.2. Compute the membership μ value of the COG;
      1.1.3. Generate a random number between 0 and 1;
      1.1.4. If random number is smaller than “fuzzy probability”;
         1.1.4.1. Current variable is set a value equal with COG;
      1.1.5. If random number is larger than “fuzzy probability”;
         1.1.5.1. Jump to step 1.2;
   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 “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 the influence of the rules will not be so big.

The Gaussian function can be written as:

$$f(x) = a \cdot e^{\frac{-(x-b)^2}{2 \cdot c^2}}$$

We selected: a = 1-*mutation_probability*, b = 0 and c = 150.The value of x increases with one for each individual generated by the algorithm. The parameters were selected such that after 500 individuals (x=500) the curve is “close enough” to

the value 0. With a population with a size of 100 after 5 generations, the influence of the fuzzy rules will be small. The maximum of this function is 1-mutation probability.

The function is further translated with a value equal with 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 also 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:

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

This number is used in the bit flip mutation fuzzy.

4.4.2.3.4 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 5.6 where the membership functions are generated automatically and their shape is not trapezoidal.

4.4.2.4 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. They are described inside the <*virtual_parameters*> tag in the input XML file:

```
<virtual_parameters>
        <parameter name="dl1_size" value="dl1_nsets*dl1_bsize*dl1_assoc"/>
</virtual_parameters>
```

Any number of virtual parameters can be defined. All the expressions possible inside constraints (see Paragraph 4.2.2) can be used here.

Virtual parameters than are defined in the XML input file can be used only as inputs inside the FCL file. Using a virtual parameter as output can not be implemented because it will be unknown how to change the individual parameters to make the virtual parameter big/small for example.

4.4.2.5 Starting FADSE with A FCL File

To start FADSE using rules the user has to specify as a command line parameter the FCL file: *./java –jar fadse.jar inputFile.xml rules_file.fcl*

4.5 Summary

This chapter introduces some advanced features integrated in FADSE. The general evolutionary/ bio-inspired algorithms can be helped through domain knowledge. Representing this domain-knowledge and influencing the course of the DSE algorithms with it is not straightforward. We have taken some previous known methods (constraints, hierarchical parameters), developed new ones (fuzzy rules as prior information) and integrated into FADSE into an original manner.

The chapter starts with the implementation of constraints in FADSE. These can be used to limit the size of the design space, to set some borders outside which individuals become infeasible. Inspired by the M3Explorer constraint interface, we developed our own and implemented new powerful features. Probably the most important ones are the expressions that can be used inside constraints.

Next we analyzed the need for hierarchical parameters. We concluded that there are many situations when parameters depend on another one. For this we have implemented new features in the XML interface so that the user can describe these relations easily. We also implemented new crossover and mutation operators that take into consideration this information.

Most likely one of the most important features present in FADSE is the possibility to influence the algorithms by expressing human readable fuzzy rules. To our knowledge we are the first to propose this, at least related to computer architecture problems. The idea is that the user can express his knowledge/expertise into a common language using linguistic terms, which will be then translated into mathematical functions and used to guide the design space exploration process. In the next chapters we will prove that this concept can lead to great results, and also we demonstrate its flexibility.

To use these fuzzy rules, we developed new mutation operators that take into consideration the information provided. We proposed two different operators: one

uses a constant probability to apply the fuzzy rules, the other one uses a Gaussian distribution of probability.

As a last improvement we included the virtual parameters, as we called them. This is an extension meant to allow a more flexible description of the rules.

More details about these features and how to use them can be found in our technical report: “An overview of the features implemented in FADSE” [73].

We would like to specify that the order of the rules from the FCL input file is not important. They are aggregated using a commutative operation.

"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 [74] [75] [76] (submitted) or presented at workshops [77].

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.

The GAP has a superscalar-like front end; the difference compared to a traditional processor appears in the configuration unit. This new configuration unit maps dynamically independent instructions, data or control flow dependent instructions into a matrix of functional units (FUs). Each column of the matrix is dynamically assigned to one architectural register. The instructions are assigned to the column whose register matches the instruction's output register. Dependencies between instructions are modeled using the rows of the matrix of FUs. Each element of the matrix contains some memory cells. These form configuration layers (they work similar to a trace cache) which are used to save configurations for repeated execution. The number of configuration layers can be: 2, 4, 8, 16, 32, or 64. More details about the GAP processor can be found in [78] and [79].

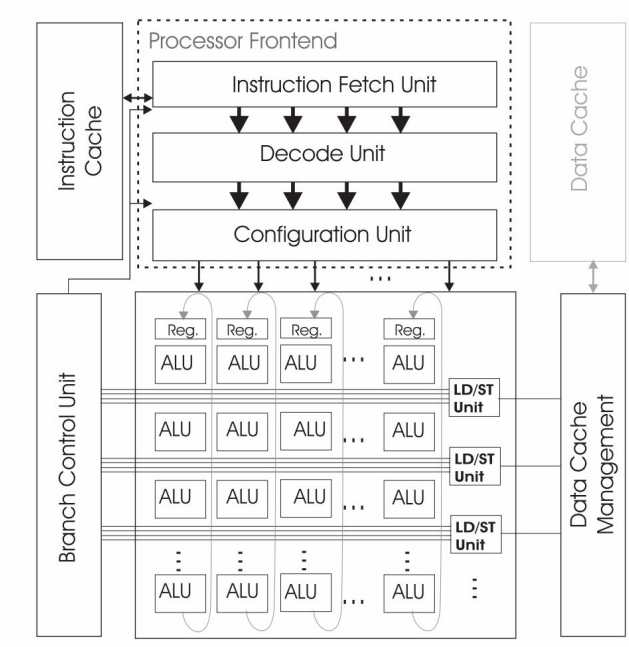


Figure 5.1-1 Architecture of the Grid ALU Processor

GAP was designed to be scalable, to use the small to large processor dies, depending on the required performance/complexity ratio required.

The number of configuration layers, of FUs, the size and the organization of the caches can be varied. The domains are presented in Table 5.1-1.The size of the space is over 1 million (1016640) possible configurations.

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}

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. GAPtimize uses information about the target architecture and profiling data, acquired from previous runs of the program, to perform feedback-directed and adaptive code-optimizations. Some of the code optimizations implemented are: predicated execution, a special scheduling technique, inlining of functions, a software-assisted replacement strategy for the configuration layers supported by code annotations called qdLRU [80], static speculation [81].

In this chapter we focus mostly on function inlining as a code optimization. In the last section we also use the replacement strategy (qdLRU) and static speculation.

Table 5.1-2 Parameter space for function inlining

Parameter	Domain
max.caller.count	{0, 1, 2, 3, ..., 100}
weight.of.caller	{0, 1, 2, 3, ..., 100}
length.of.function	{0, 1, 2, 3, ..., 10000}
insns.per.caller	{0, 1, 2, 3, ..., 200}

The function inlining technique replaces function calls with copies of the body of the called function. More details about the advantages brought by this technique in GAP can be found in [74].

The problem in function inlining is how to choose the right function callers that will be replaced. The algorithm, which chooses the function callers, works as follows: not more than *max.caller.count* callers are inlined (see Table 5.1-2). Data is collected from a reference run and all of the callers must be executed at least *weight.of.caller* times. The called function must not have more than *length.of.function* static instructions. A classification number for each caller must not be larger than *insns.per.caller* (see our article [74] for more details).

The size of the space of GAPtimize only with function inlining is $2*10^{10}$.

The idea behind static speculation is to use spare functional units of the GAP to speculatively calculate instructions before it is known if they should be executed in the further program flow or not. In other words speculation concerning control flow edges is done statically by the post-link optimizer GAPtimize.

Due to hardware restrictions there is quite a small number of configuration layers in a GAP architecture. It is important to use them wisely as their impact on performance can be high because instructions found in them do not have to pass the processor front-end to be re-executed. In experiments it was shown that thrashing can dramatically reduce the performance of least recently used (LRU) as replacement policy. QdLRU is an extended version of LRU that can be directed with flags in the program code to make it less prone to thrashing. If QdLRU is disabled then LRU is used as replacement strategy.

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.

5.1.2.1 CPI and CPRI

It is a common method to give the processor performance in terms of instructions per cycle (IPC). Since we want to have both objectives minimized, we inverted this metric and expressed the performance as 1/IPC = cycles per instruction (CPI).

The GAP simulator, we are using for the DSE process, computes the number of clock cycles for the simulation and the number of dynamic instructions executed. These are used to compute the CPI. This metric is comparable because the number of instructions remains the same for all the runs (for the same benchmark), regardless of the processor configuration.

When running with GAPtimize the above statement is not true. GAPtimize can change the number of instructions inside a program, thus CPI is not comparable anymore. To avoid such problems, we are running all the benchmarks once, obtain the number of dynamic instructions (called reference instructions) and then use them as reference in the metric computation. We call this metric: clocks per reference instruction (CPRI).

5.1.2.2 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.

To compute the complexity (H) we divide it into several components: front-end (H_{front}), the complexity of the array of ALUs and load store units (H_{ALUs}+H_{LSUs}), the instruction cache (H_{ICache}) and other components which are not varied by us ($H_{constant}$). So the complexity (H) can be written as:

$$H = H_{front} + H_{ALUs} + H_{LSUs} + H_{ICache} + H_{constant}$$

We removed the complexities not changed by our parameters (H_{front} and $H_{constant}$) from the equation because we only need this metric to compare different configurations.

We decided to express everything in terms of size of an ALU (so ALU has a size of $h_{ALU} = 1$). We analyzed results from a couple of articles [82] [83] [84] and obtained the results (relative to h_{ALU}) presented in Table 5.1-3.

Table 5.1-3 Hardware complexities relative to ALU size

Data Source	32 bit register	64 kb ICache
McPat for Alpha 21364 [82]	0.031	15.678
HotSpot Alpha 21264 [84]	0.008	16.767
Gupta et al. [83]	0.017	17.131
Average	0.019	16.525

From Table 5.1-3 we extract the size for a register $h_r = 0.02$ (rounded from 0.019). We set the size of a configuration layer h_l to 0.02. We concluded that a 64KB ICache entry is 16.5 times larger than one ALU. With CACTI [85] we compute the size of a cache similar with the one used in the above mentioned articles and obtain an area of 5.53 mm^2. We will compute the size of the cache with CACTI automatically and then multiply this value with $\frac{16.5}{5.53mm^2} \approx 3 * \frac{1}{mm^2}$ to obtain a value relative to the ALU.

For the LSU unit we obtain results from McPAT and the floor plan and set a complexity: $h_{LSU} = 3.5$.

To compute H_{ALUs} and H_{LSUs} we use the following equations:

$$H_{ALUs} = (C_c * h_r + C_r * C_c * h_{ALU}) + C_r * C_c * C_l * h_l$$

$$H_{LSUs} = C_r * h_{LSU} + C_r * C_l * h_l$$

where C_c is the number of columns, C_r is the number of rows, and C_l is the number of configuration layers of the array of FUs.

More details about this metric can be found in our article presented at HPCS 2011 International Conference [74].

5.2 Related Work

There is no prior work on the approximation of the hardware complexity of the GAP. The presented DSE is also the first multi-objective one for this processor.

In the following paragraphs we compare several algorithms using GAP and GAPtimize. In [31] a comparison between different multi-objective optimization algorithms on the Low-Power Processor Use Case (Superscalar Processor SP2) delivered by STMicroelectronics-China simulator is presented.

The algorithms used are: NSGA-II, MOGA-II, MOSA, ES, DPSO [27], MFGA and APRS.

The authors evaluate the configurations on a single benchmark: *164.gzip* (based on the *gzip* benchmark). Their simulator has 11 parameters but the authors use only 7 parameters reducing the design space to 9216 possible configurations. This allows them to perform an exhaustive search. By having all the possible configurations evaluated, they can use metrics that require the true Pareto front for evaluation. The metric used is Average Distance from Reference Set (see Chapter 2.6.2.4). The conclusion is that NSGA-II obtains the best results.

In the article describing Magellan [49] a set of single objective algorithms are compared (genetic algorithms, ant colony optimization, hill climbing, random search, etc.) against an exhaustive simulation. They reduce the search space greatly be fixing many parameters of the used simulator, thus allowing them to perform an exhaustive search. The authors conclude that a genetic algorithm can reach a performance worse with 0.1% than the optimal solution but 23000 times faster. The drawback of this work is that it was not extended to a multi-objective problem.

Most of the articles found in the literature compare algorithms on synthetic problems. The closest article to our work is [21] where SMSPO is compared with NSGA-II and SPEA2. The article concludes that SMPSO clearly performs better. No clear conclusion can be drawn from the comparison of NSGA-II and SPEA2. The same conclusion is reached in [14] where a comparison between SPEA2 and NSGA-II is performed.

Regarding DSE on GAP a manual non-exhaustive manual exploration has been performed in [78] on 14 benchmarks from the MiBench [86] suite. The authors fixed the cache at a size of 8kB. They suggest as a rule of thumb to use the same number of lines and columns for the matrix of FUs up to 16x16. For the largest array they use 16 columns and 32 rows. We are using their results as a reference for comparison.

5.3 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 [87]. He has performed an extended DSE exploration and we try to see if better results can be obtained automatically.

5.3.1 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 [13]. Bit flip mutation, single point crossover and the binary tournament selection, proposed in [13], 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 et al. 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.

5.3.2 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 5.3-1). 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.

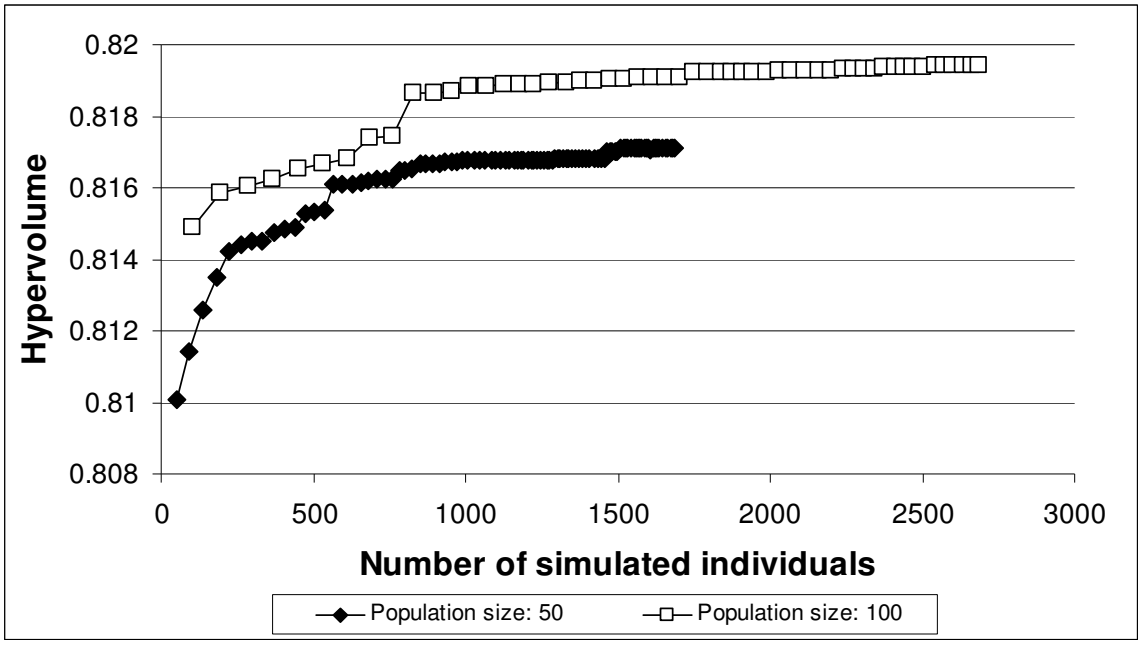


Figure 5.3-1 Average hypervolume comparison between runs with a population size of 100 and runs with a population size of 50

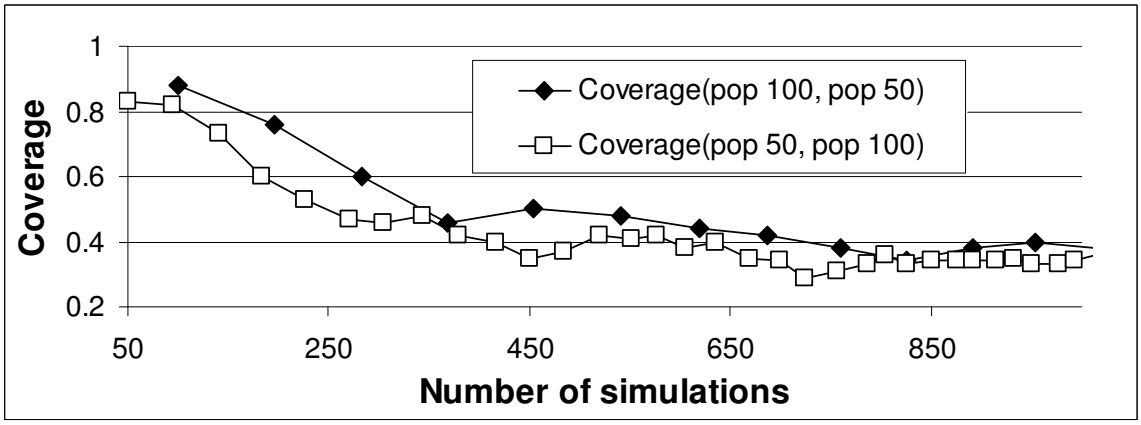


Figure 5.3-2 Coverage comparison between a DSE process with a population of 100 and another one with a population of 50

We decided to compare the two runs using the coverage metric (see Paragraph 2.6.3.4). According to the coverage (see Figure 5.3-2), 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 5.3-3 we compare the results obtained by FADSE against the manually obtained results in [78]. In Figure 5.3-4 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.

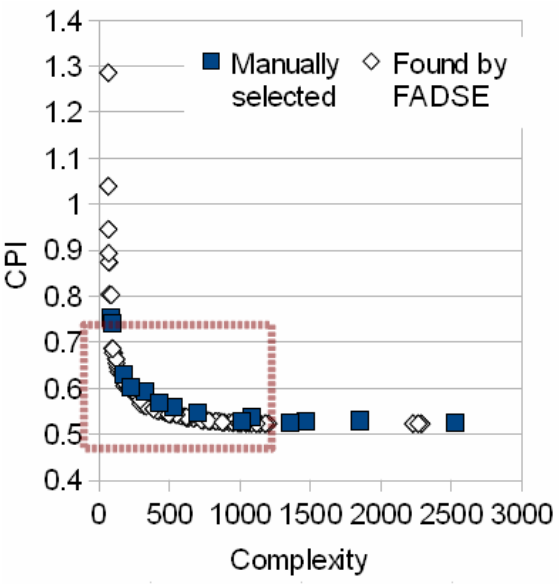


Figure 5.3-3 Total result space

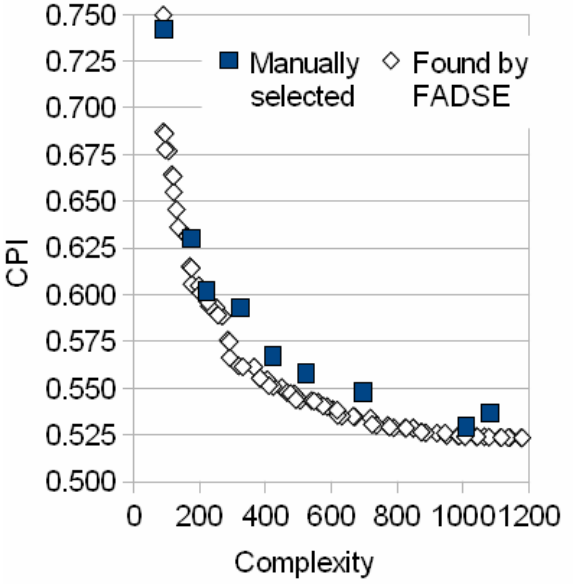


Figure 5.3-4 Zoom on most interesting area (highlighted box in Figure 5.3-3)

In Figure 5.3-3 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 5.3-5 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.

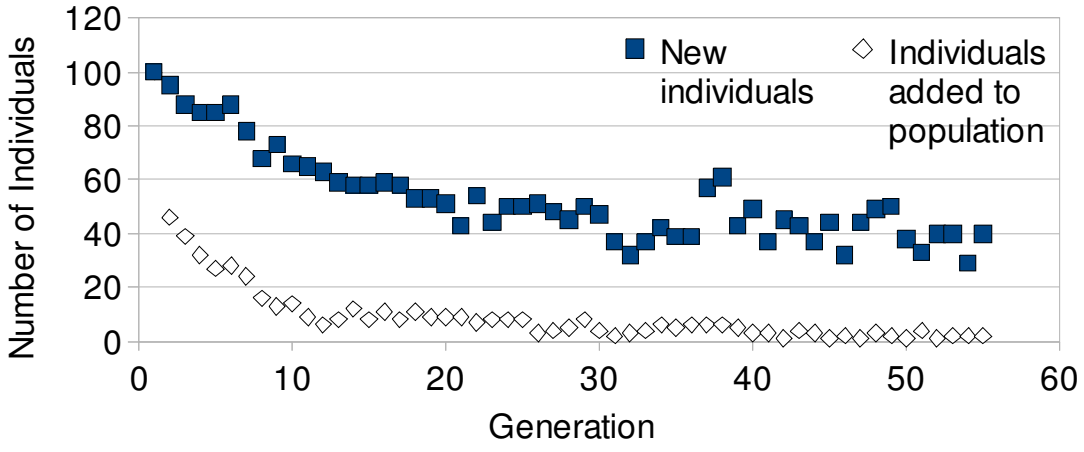


Figure 5.3-5 Comparison between the number of newly generated individuals (offspring) and the number of them that actually reach the next generation

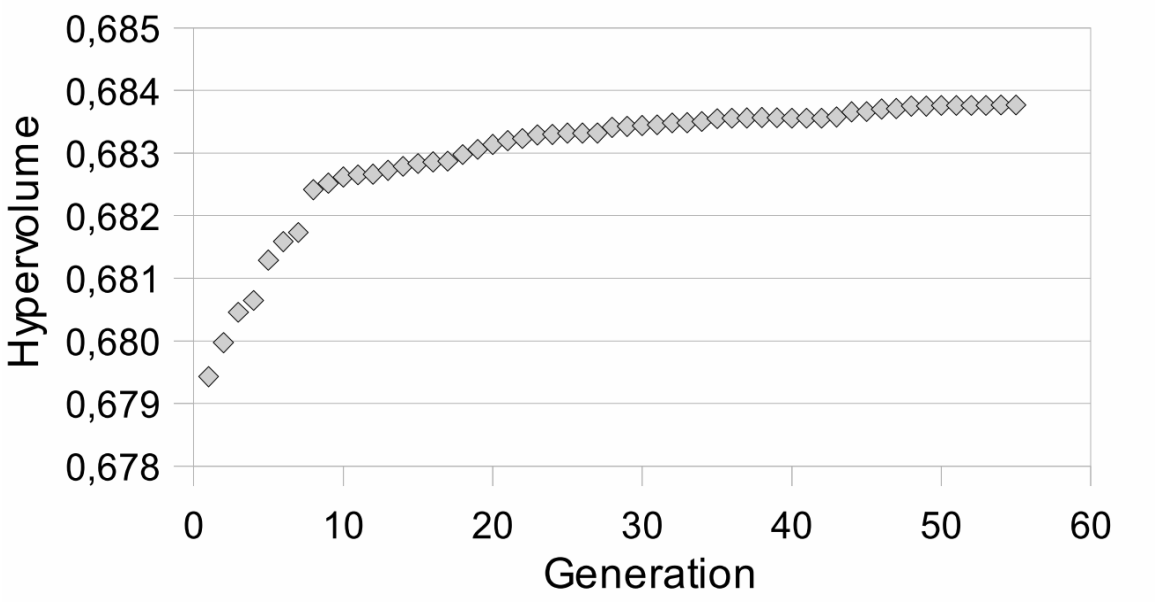


Figure 5.3-6 Evolution of the hypervolume value over the generations

Figure 5.3-5 is correlated with Figure 5.3-6 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.

5.3.3 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.

5.4 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 (see Paragraph 5.1).

5.4.1 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. $1.8 *10^{13}$.

5.4.2 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 thus optimize 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.

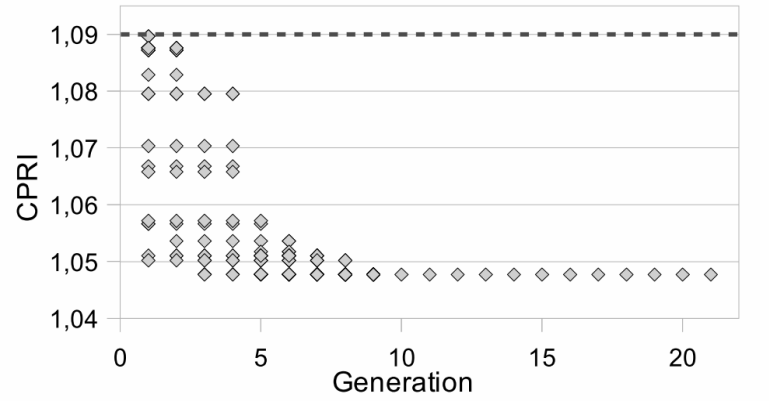


Figure 5.4-1 DSE of the inlining parameters for 10 benchmarks, executed on GAP with 12x12x8 array and 8 kB instruction cache. The dotted grey line is the CPRI without inlining.

Next, the number of benchmarks was increased to 10, but the hardware remained fixed, so we still have single objective optimization. In Figure 5.4-1 the obtained results are depicted. 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.

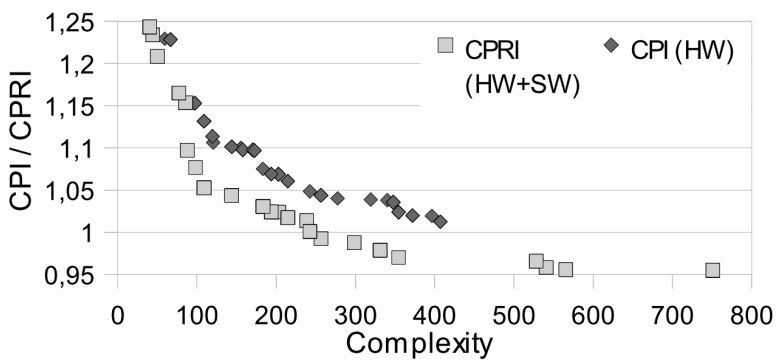


Figure 5.4-2 DSE of inlining and hardware parameters for 10 benchmarks, executed on GAP with NxNx8 array and 8 kb instruction cache

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 5.4-2 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.

5.4.3 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.

5.5 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" [75].

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) – see chapters 2.3 and 2.4 for more details.

5.5.1 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 [21]:

- 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 5.4.1 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.

5.5.2 Results

5.5.2.1 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 5.5-1). 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.

After comparing SMPSO with SPEA2 (see Figure 5.5-2) we can see that SPEA2 finds worse results than the ones obtained by SMPSO throughout the DSE process. A small recovery from SPEA2 is seen around generation 10 but then SMPSO starts gaining again.

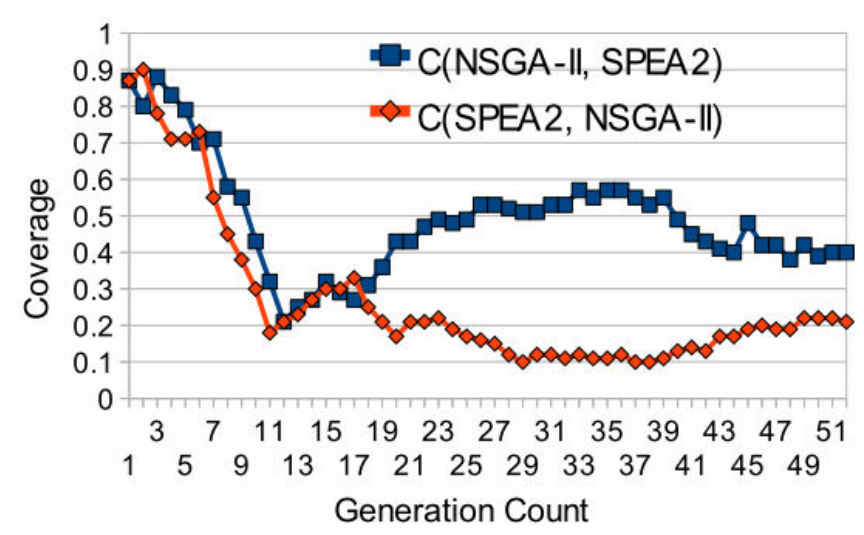


Figure 5.5-1 Coverage comparison between NSGA-II and SPEA2

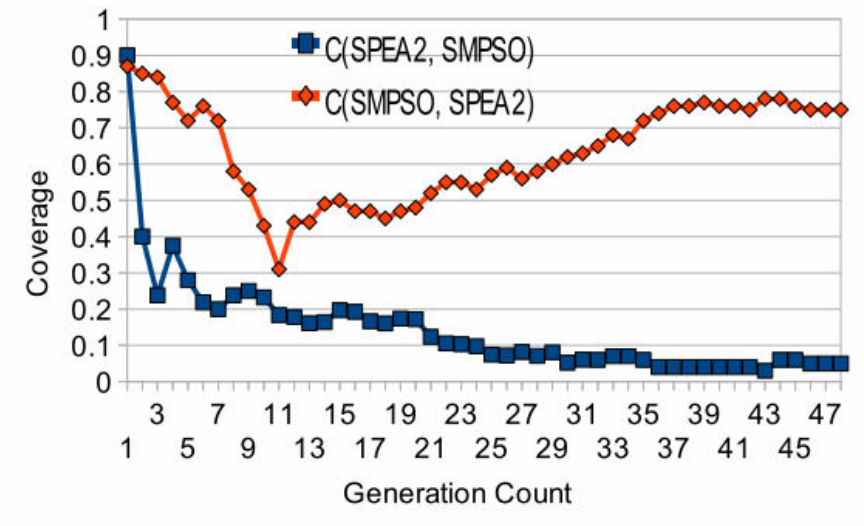


Figure 5.5-2 Coverage comparison between SPEA2 and SMPSO

Next we compared NSGA-II and SMPSO. In Figure 5.5-3 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.

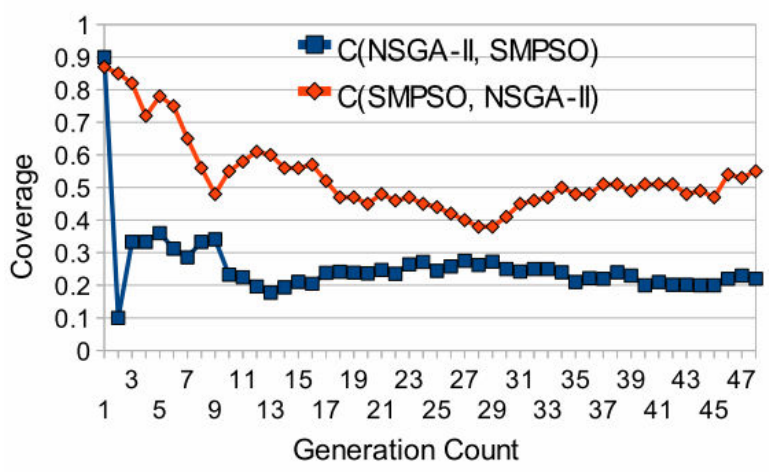


Figure 5.5-3 Coverage comparison between NSGA-II and SMSPO

We compared the algorithms from the perspective of a different metric: hypervolumne. The evolution of the hypervolume over the generations is shown in Figure 5.5-4. The hypervolume gives information about the convergence of the algorithm. When the same reference point is used (see 2.6.3.2), 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 5.5-4 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 5.5-5). 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.

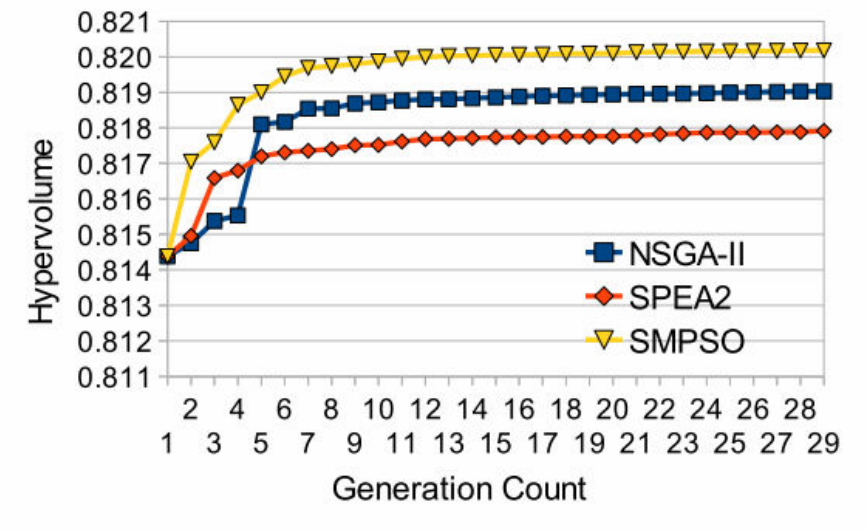


Figure 5.5-4 Hypervolume comparison between NSGA-II, SPEA2 and SMPSO

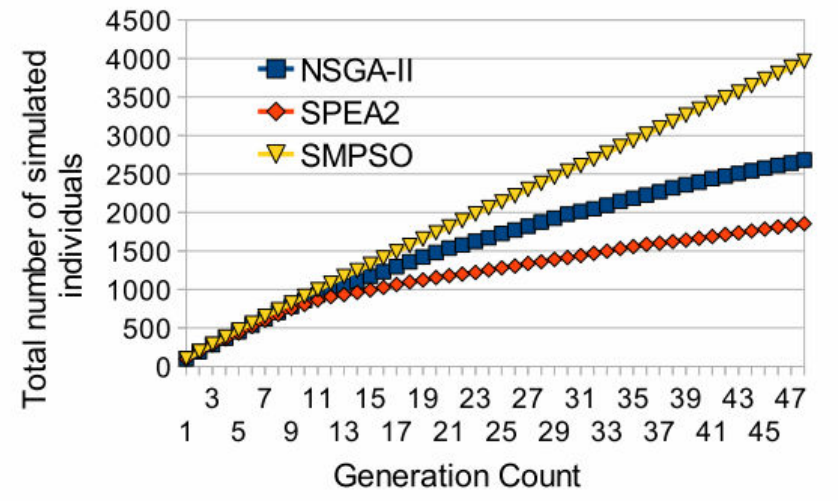


Figure 5.5-5 This figure shows how many individuals were simulated to reach a certain generation.

With this in mind we decided to compare the hypervolume from this point of view. In Figure 5.5-6, 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 5.5-5 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 2.3.5. 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.

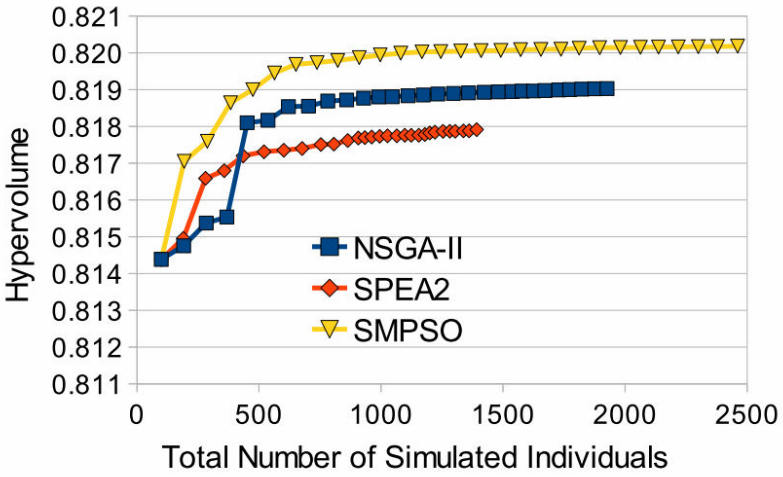


Figure 5.5-6 Hypervolume comparison of the three selected algorithms against the total number of evaluated designs

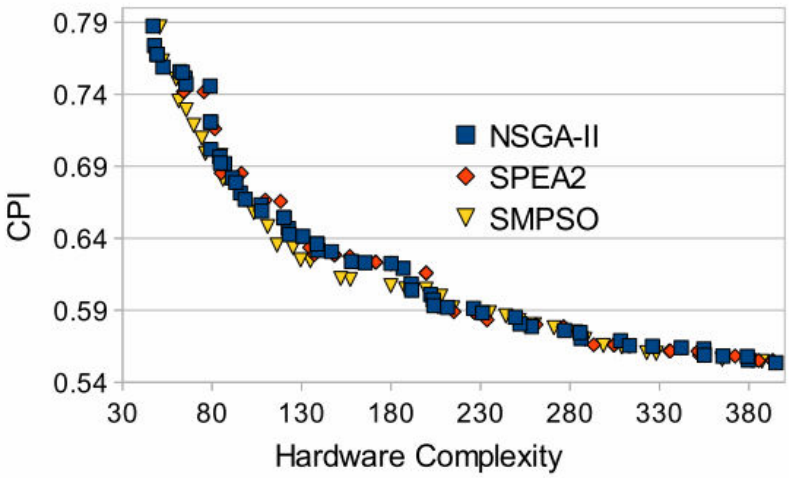


Figure 5.5-7 Section of the Pareto front approximations obtained by each algorithm after 50 generations

We compared the Pareto fronts approximation found by the algorithms (see Figure 5.5-7). 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 2.6.3.3) to see how much of the space is really dominated by a single algorithm. The results can be seen in Figure 5.5-8. 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.

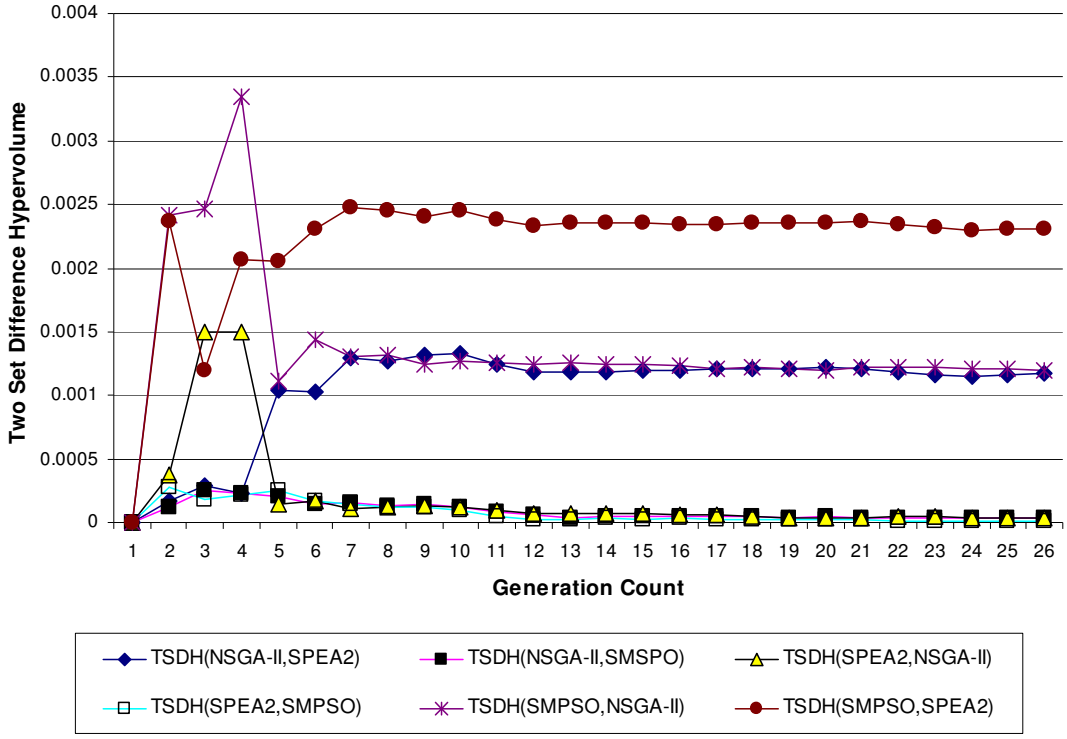


Figure 5.5-8 Comparison between the algorithms using the Two Set Difference Hypervolume (TSDH) metric

5.5.2.2 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 5.5-9). 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 5.5-10). SMPSO has the best results, but the difference is not that big, as we have seen between SPEA2 and NSGA-II.

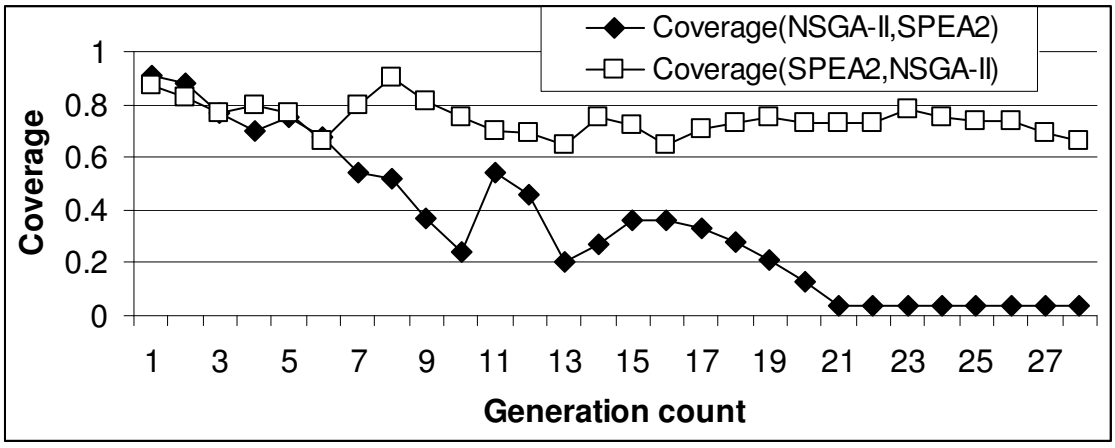


Figure 5.5-9 Coverage comparison between DSE runs with NSGA-II and SPEA2

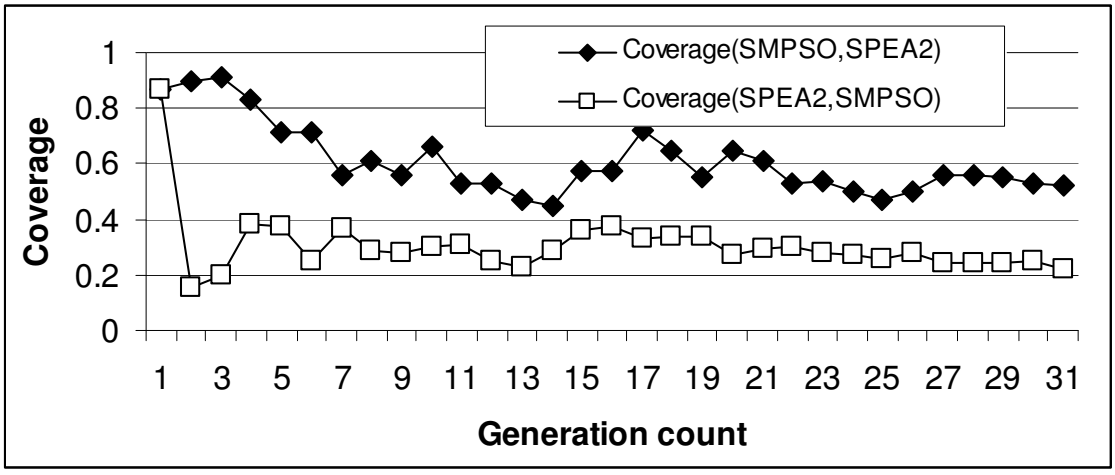


Figure 5.5-10 Coverage comparison between DSE runs with SPEA2 and SMPSO

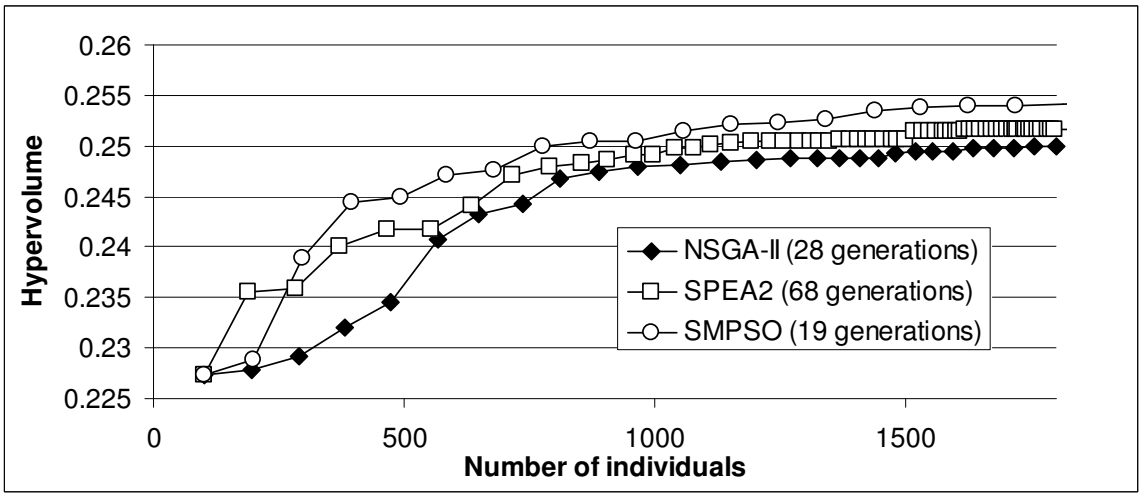


Figure 5.5-11 Hypervolume comparison between NSGA-II, SPEA2 and SMPSO considering the number of simulated individuals

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 5.5-11). 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 5.5-12). There are not much individuals from other algorithms except SMPSO in the figure because they are overlapped. The algorithms obtain practically the same results.

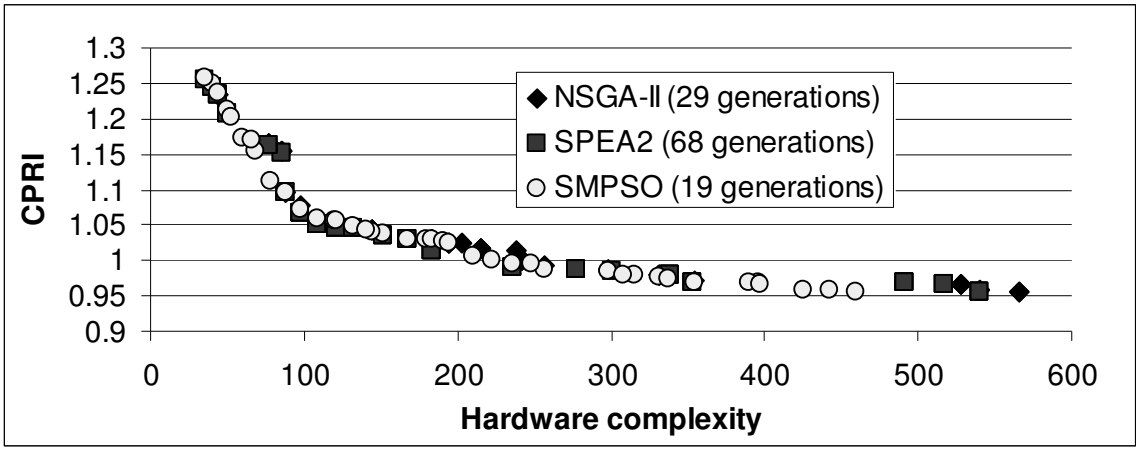


Figure 5.5-12 Final Pareto front approximations obtained by NSGA-II, SPEA2 and SMPSO after 1800 simulations

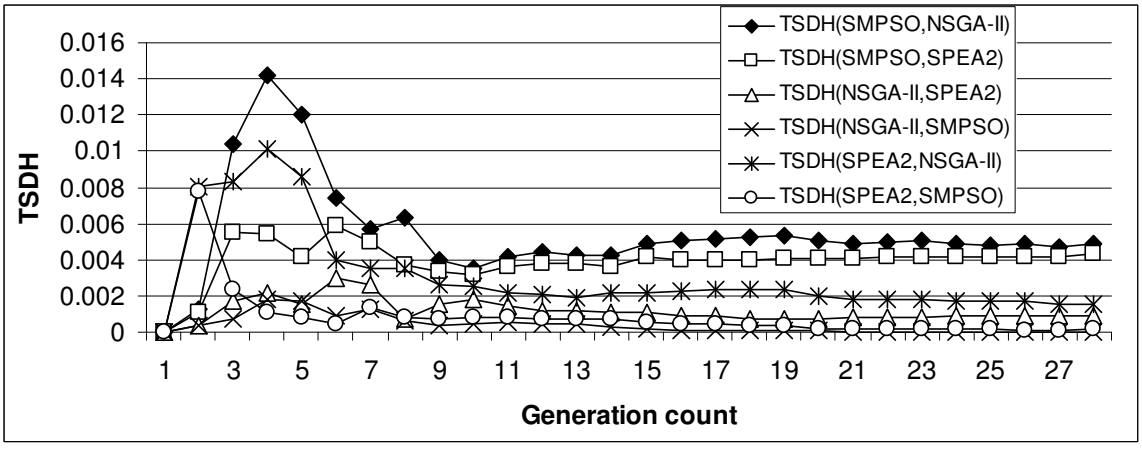


Figure 5.5-13 Two set difference hypervolume (TSDH) comparison between the algorithms

Finally we compare the three algorithms (two by two) using the two set difference hypervolume metric – TSDH – (see Figure 5.5-13). 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.

5.5.3 Conclusions

Analyzing the Pareto fronts approximation 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 [21]. 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 [20]. 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.

5.6 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 extract to 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" [76].

5.6.1 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.

The problem is how to extract information from points and represent this information as rules. The purpose is to obtain rules that will change an individual and make it better. The approach adopted was to extract the good individuals and use them as basis for rule generation. The individuals were classified as "good" or "perfect". The *perfect* ones are the individuals closest to the Pareto front approximation found in the previous exploration. The *good* individuals are the rest. Since the obtained Pareto front approximation is not continuous, imaginary points were generated through interpolation techniques. This is necessary because, in areas with no points on the Pareto front, the individuals that need to be classified might appear to be at a great distance, when in fact they are close to the interpolated surface. One third of the individuals are classified as *perfect*, the rest are classified as *good*. This ratio is obtained by adjusting the maximum distance from the Pareto front at which an individual has to reside to be classified as *perfect*. The next step was to use the data mining tool WEKA [88]. This tool automatically constructs a decision tree with parameters as nodes inside the tree and our classes (*perfect*, *good*) as leafs. From this tree, rules are extracted easily. More details about this can be found in our article [76].

The generated rules usually do not form a complete rule base (see 4.4.1.2.2). This means that there will be individuals that are not matched by any of the rules. To increase the probability to apply the rules, different techniques have been tried. One of them, that lead to good results, was to introduce additional rules. These rules are only used to move the value of a parameter from an interval, where no rule was defined, to an interval covered by the initial rules. This lead to an increase with 10% of generated *perfect* individuals.

Using this approach the rules have sharp edges (are not fuzzy in the true sense). Because of this, the random defuzzifier had to be used (see 4.4.2.3.4). If the center of gravity defuzzification method would have been used, the output of the rules would always be the same.

The jFuzzyLibrary does not have an exact representation of the membership function. The values on the core (see 4.4.1 membership function terminology) are not always 1 (as described in the FCL file), but close to 1. Thus, the threshold value implemented in the random defuzzifier had to be modified. We tried with different values for the threshold (membership threshold - MTH): 0, 0.2, 0.4, 0.6 and 0.8 (see Figure 5.6-1).

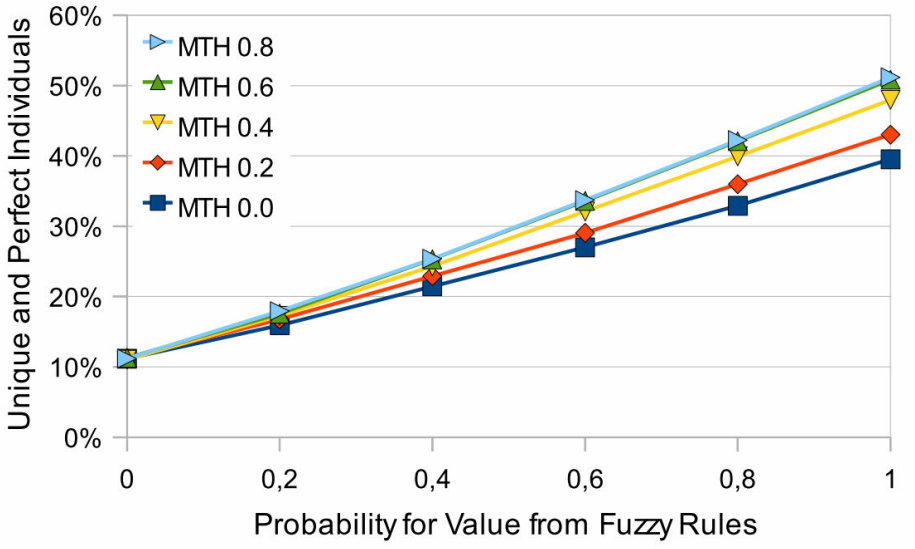


Figure 5.6-1 Ratio of perfect and unique generated individuals in different situations

Concretely 2833 individuals (2385 were unique individuals) obtained during the exploration from Chapter 5.3 were gathered. A distance of 0.018 was selected so that 35% percent of the total individuals are classified as *perfect* (994 individuals). The data mining tool WEKA (as described before) obtains a tree with 28 leaves, out of which 10 classify the *perfect* individuals. The obtained tree is further reduced to 5 leaves by selecting only the most important paths (the ones that classify the most individuals). This tree is able to classify over 80% of the individuals correctly. Rules are obtained from this tree (20 rules). To these rules, three rules were added to provide full coverage of the parameters.

The obtained rule set was further evaluated. For this test we have generated 1000 individuals randomly and applied the fuzzy rules with different probabilities over 50 iterations. In Figure 5.6-1 the averages are presented. We can see that if we do not apply the fuzzy rules at all, around 11% of the total individuals are classified as *perfect* (if duplicates are generated by the algorithm they are eliminated). The higher the chance to apply the rules, the higher is the percentage of *perfect* individuals. The figure also tells us that when the MTH value gets higher we have better individuals. For a MTH of over 0.99 the random defuzzifier stops working (not shown in the figure) because it does not find points with such a high membership. We decided to set the MTH to 0.8 in our experiments.

We performed tests with both implementations for the fuzzy mutation (see 4.4.2.3) 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.

5.6.2 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 5.6.1 and extracted the rules.

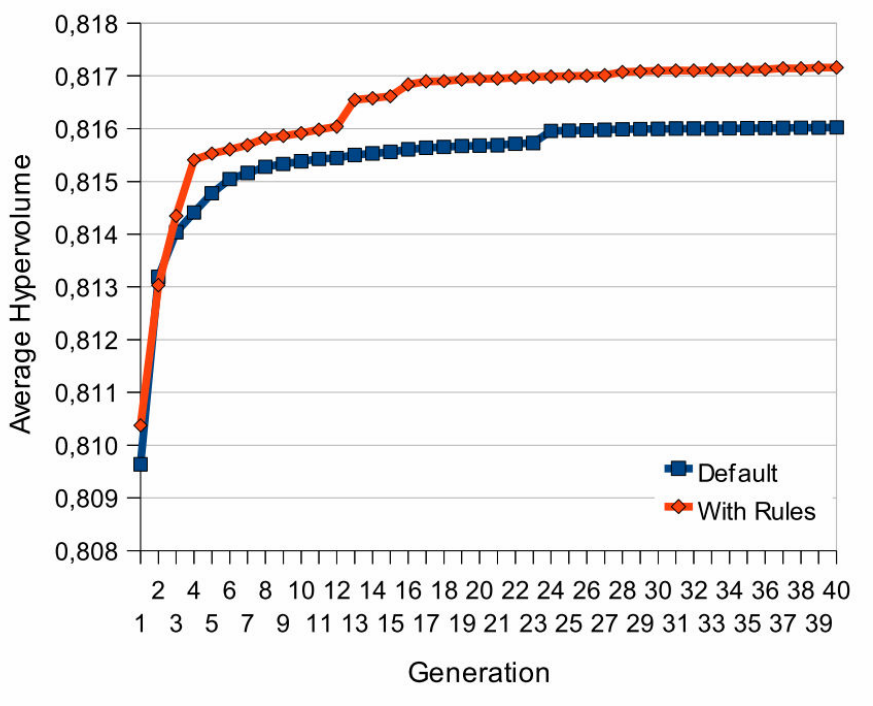


Figure 5.6-2 Special to general experiment

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 5.6-2 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. In this situation, we have used the results from Chapter 5.6.1 as input data. 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.

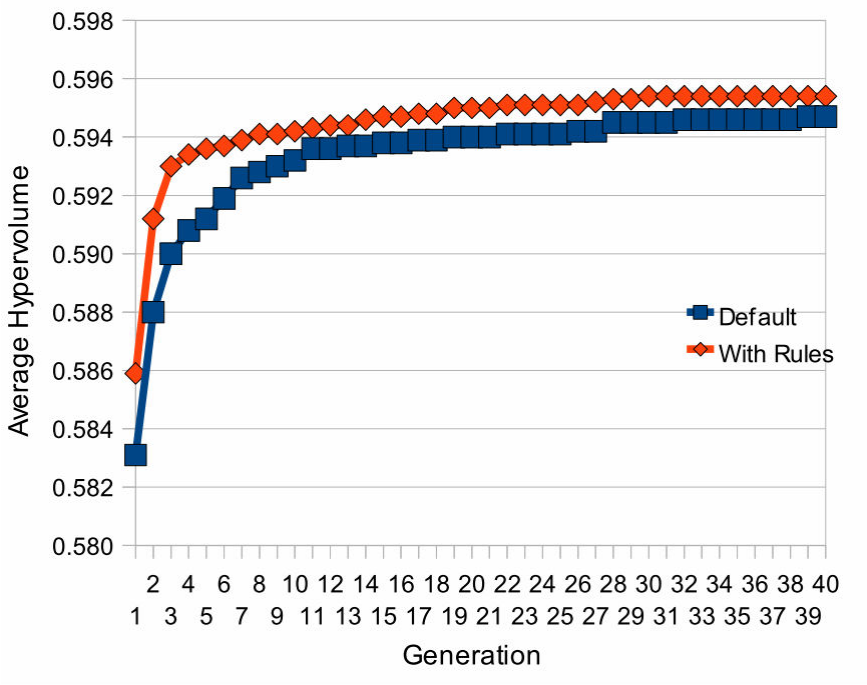


Figure 5.6-3 Hypervolume JPEG

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

The average results for JPEG are shown in Figure 5.6-3, for *Rijndael* in Figure 5.6-4. 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 5.6-3 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 6.3 we observed that runs with constant probability work better when the rules do not cover so much of the available parameters.

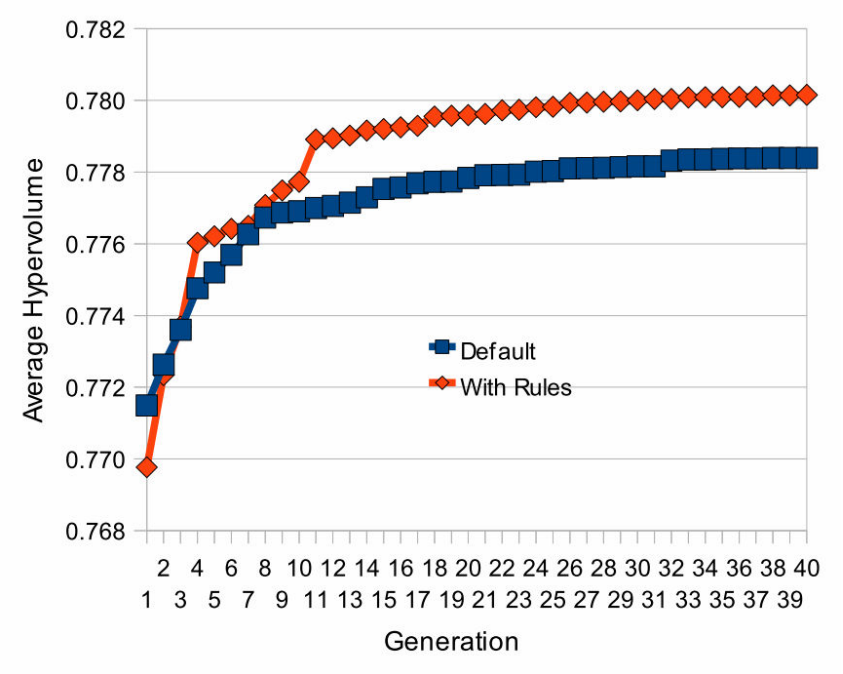


Figure 5.6-4 Hypervolume Rijndael

5.6.3 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.

5.7 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 in Chapter 5.1. 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.

5.7.1 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 Chapter 4.3.3. We ran the experiment for 40 generations.

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

5.7.2 Results

The preliminary results obtained for quick sort are shown in Figure 5.7-1. It can be seen that the run with the hierarchical parameters obtained better results than the run without this information.

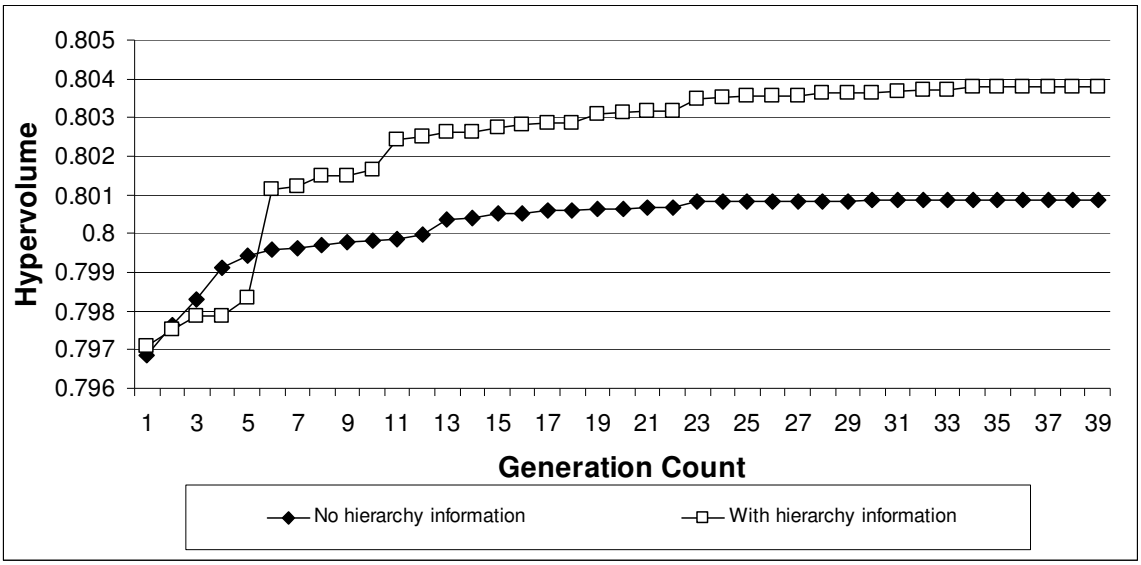


Figure 5.7-1 Hypervolume comparison between the run that considered the information about hierarchical parameters and the one that did not

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.

5.8 Summary

In this chapter we focused on the GAP architecture alongside with the code optimizer developed for it: GAPtimize. Most of the features presented in Chapter 3 were developed as a requirement of these experiments (we benefited from them for other experiments too): distributed evaluation, database integration, reliability concerns.

Especially for this exploration GAP has been developed to output a second objective beside CPI: hardware complexity.

We ran FADSE with GAP on several benchmarks from the MiBench suite and proved that FADSE can cope with a large design space and it can adapt to the GAP simulator.

In our first experiment we compared the results obtained by a human designer (Dr. Basher Shehan who actually implemented the simulator for the GAP architecture) with the results obtained by FADSE. We proved that the human designer might be biased and thus influence the results in a bad way. We managed to find with FADSE configurations with half the complexity at the same CPI, compared with the ones found by a human designer. We were also able to disproof the rule of thumb used by the designer in his experiment.

Next challenge was to use FADSE for hardware and code optimizations at the same time. This meant a huge design space and also different domains involved. FADSE was able to cope with this too and obtained very good results.

Up to this point we have focused only of the NSGA-II algorithm. We decided to compare other algorithms on GAP and GAP with GAPtimize. For this we selected another genetic algorithm (SPEA2) and a bio-inspired one: SMPSO. All these three algorithms are included in FADSE and we changed them to run in a distributed manner. First we compared all three algorithms while trying to optimize the GAP architecture. The particle swarm optimization lead to great results both in terms of convergence and quality of results. For this experiment NSGA-II performed better than the SPEA2 algorithm. Next we moved to the GAP with GAPtimize combination. Again the PSO algorithm emerged as the winner both in terms of quality of results and convergence speed. The difference is that in this test SPEA2 performed better than NSGA-II. A conclusion drawn from this experiment was that the metrics we were using might be misleading. We observed that the coverage metric showed great differences between the algorithms but in reality (analyzing the Pareto front approximations found) the differences between the results are minimal.

In our next experiment, we used the interface developed for fuzzy rules in a slightly different way. Now the rules are not provided by the human expert but are generated automatically from previous explorations. We identified two situations when this might be interesting: an experiment could be run on a single short benchmark that will run very fast. From this experiment, some knowledge could we be extracted with the methods we have used/proposed and rules could be generated automatically. These rules could then be used to accelerate the DSE process and help it find good results faster. The second possible situation would be in the case of a company that already did its DSE on this architecture, but a new processor, similar with the existing one, has to be developed. Instead of starting from scratch it could use what has been learnt before and accelerate the DSE for the new design. For this, we presented a method to extract automatically rules from the previous obtained results. We tested this idea on GAP and showed that it can improve the results considerably. Important gains were obtained both in terms of convergence speed (50% faster for the same results) and also the quality of results: the results obtained with the rules are never obtained without the rules in the same amount of allocated time.

Finally, results were obtained using the hierarchical parameters presented in Chapter 4.3. The results obtained are preliminary and improvements need to be made in FADSE to use the provided information in the best possible manner.

"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 [6] and some subsequent articles [89][90]. 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 [91] is a cycle accurate processor simulator based on SimpleScalar 3.0d [92]. It allows multi-threaded [93] micro-architectural simulation. The target architecture is a superscalar Alpha AXP 21264.

M-SIM integrates the Wattch framework [94] for power consumption estimation. It is able to run benchmarks from suites like: SPEC 2000 [95], 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 [90]. 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 [89]. 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 Related Work

Manual design space exploration performed by Árpád Gellért et al. using the Alpha architecture implemented in the M-SIM simulator enhanced with the SLVP scheme showed that the prediction structure can increase performance substantially both in terms of speed and energy consumption. In addition, it proved that with the SLVP the cache size can be lowered (less energy) retaining the same speed of the architecture.

Regarding the SLVP, Lipasti et al in [96] were the first to introduce Load Value Prediction as a new data-speculative micro-architectural technique. They are exploiting the concept of value locality of the load instructions. Other authors that addressed the load value prediction are [97] [98] [99]. Power related issues regarding the value prediction scheme are addressed in [100] [101]. Different value predictors like the stride-, context- and perceptron-based, have been proposed in [3], [102] for register centric value prediction. Selective value prediction was also proposed in [103]

6.3 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 [104].

6.3.1 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. We decided to do this for several reasons:

- FADSE has been thoroughly tested for minimization problems
- Some of the metrics we have implemented in FADSE are for minimization problems
- obtained Pareto front approximations are more easy to see this way

In our charts we have used the following metrics (besides the classical ones):

Equation 6-1 Relative CPI reduction

$$CPI_{reduction} = \frac{CPI_{base} - CPI_{improved}}{CPI_{base}} \cdot 100\ [\%]$$

In Equation 6-1 CPI_{base} is the number of cycles per instruction obtained with the base architecture (see Table 6.1-1). $CPI_{improved}$ is the number of cycles per instruction obtained with the current configuration.

For the energy reduction we have used the following equation:

Equation 6-2 Relative energy reduction

$$E_{reduction} = \frac{E_{base} - E_{improved}}{E_{base}} \cdot 100\ [\%]$$

where, the same as before, E_{base} and $E_{improved}$ are the energy consumptions of the baseline and our improved architectures, respectively. Positive values of these metrics mean an improvement over the baseline architecture.

The energy used in Equation 6-2 is an average value which is computed using Equation 6-3 [measured in $W \cdot cycles$] where N is the number of benchmarks, E_i is the total or per unit energy computed for benchmark i and C_i is the total number of cycles executed within benchmark i.

Equation 6-3 Weighted average energy consumption

$$E_{Mean} = \frac{\sum_{i=1}^{N} E_i \cdot C_i}{\sum_{i=1}^{N} C_i}$$

Since the M-SIM simulator outputs power as an objective, we need to obtain the energy used in Equation 6-3. We are using Equation 6-4 where P_{mean} is computed using Equation 6-5 and T is the simulation time in cycles.

Equation 6-4 Energy consumption

$$E = P_{Mean} \cdot T$$

Equation 6-5 Instantaneous average power consumption

$$P_{Mean} = \frac{\int_0^T P(t) \cdot dt}{T}$$

The power modeling methodology used by the simulator is presented in more detail in [94].

For FADSE, we used the following parameters: population size was set to 100 as recommended in [13]. 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 [13]). For selection the binary tournament selection operator described in [13] was used. The mutation operator was changed for the runs with fuzzy information as described in 4.4.2.3. 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.

6.3.1.1 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 6.3-1, the relative CPI reduction (computed using Equation 6-1) 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 6.3-2 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.

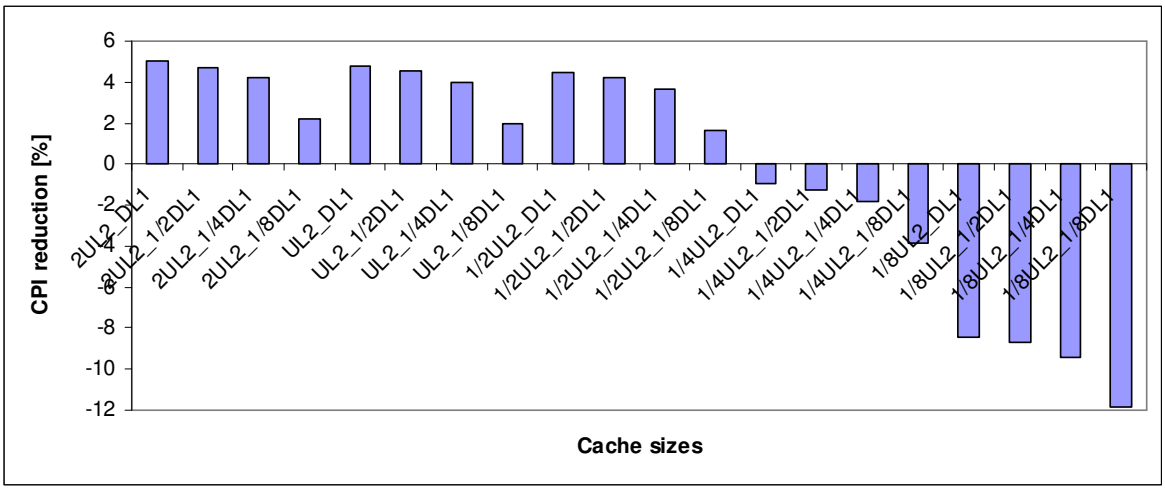


Figure 6.3-1 Relative CPI reduction reported to UL2_DL1 without SLVP as baseline

The same idea is used when we are looking at the energy reduction (see Figure 6.3-2). 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).

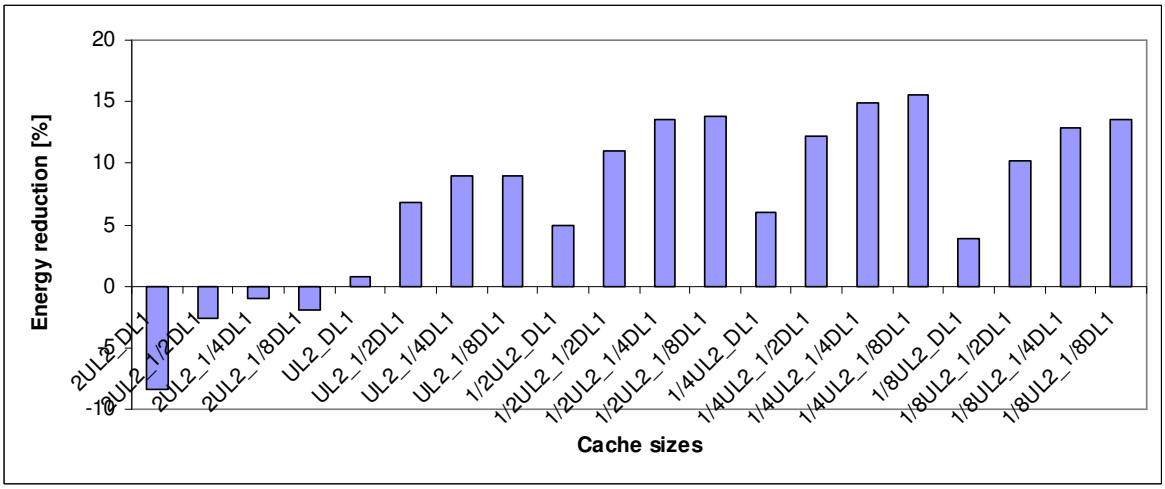


Figure 6.3-2 Relative energy reduction reported to UL2_DL1 without SLVP as baseline

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.

6.3.1.2 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 ≥ DL1_bsize
UL2_bsize ≥ 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 6.3.2) 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 (see Paragraph 3.6.1 about how this was achieved).

6.3.1.3 Running With an Initial Population

FADSE was started with some good configurations inserted in the initial population. The relaxed borders presented in section 6.3.1.2 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 6.3.1.1), and inserted them in the initial population. From Figure 6.3-1 and Figure 6.3-2 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.

6.3.1.4 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 4.4.2.4).

The Mamdani inference system was used (as described in 4.4.1.2.5).

6.3.2 Results

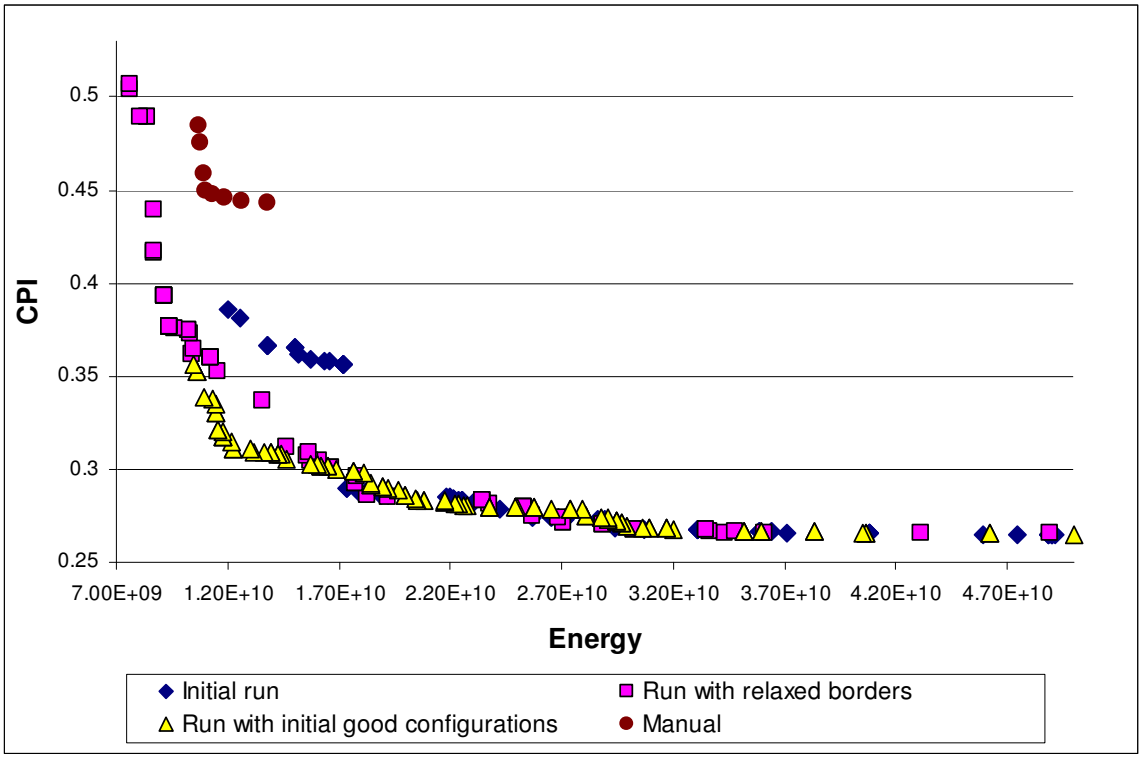


Figure 6.3-3 Pareto fronts comparison of the first runs

In Figure 6.3-3 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 6.3.1.2).

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 6.3-3 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.

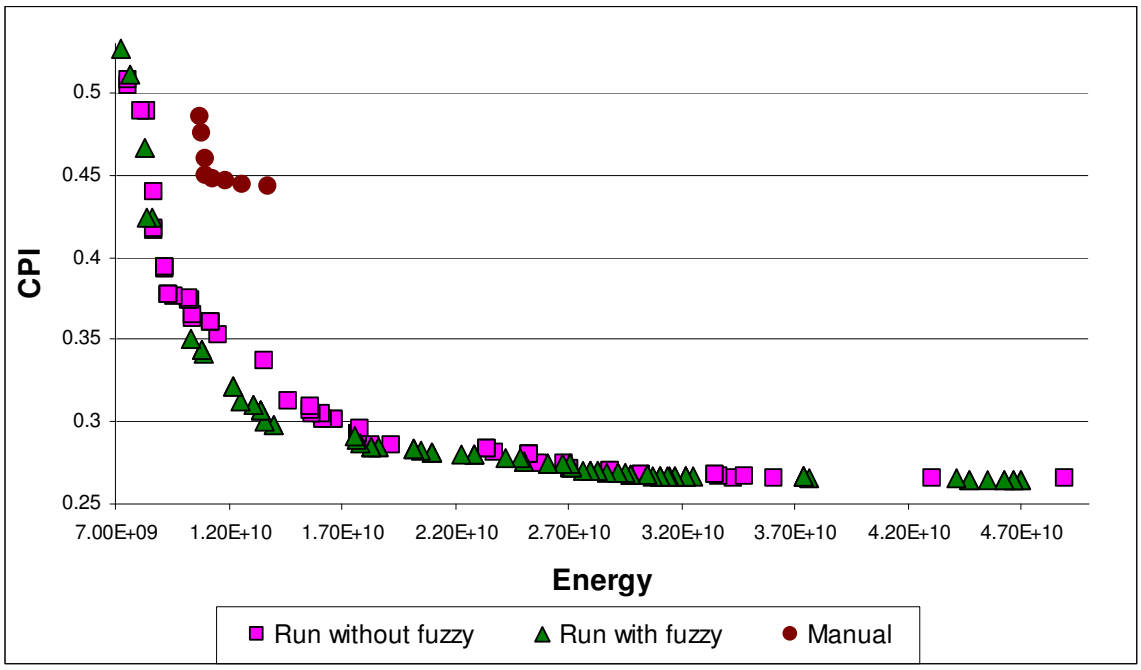


Figure 6.3-4 Pareto front comparisons between the run with fuzzy rules and the run with relaxed borders

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 6.3-4, 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 further compared the two runs using the coverage metric (see Figure 6.3-5). This new comparison confirms the superiority of the run with fuzzy information over the generations. It can be observed that during the entire DSE process, the run with fuzzy information finds better results.

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.

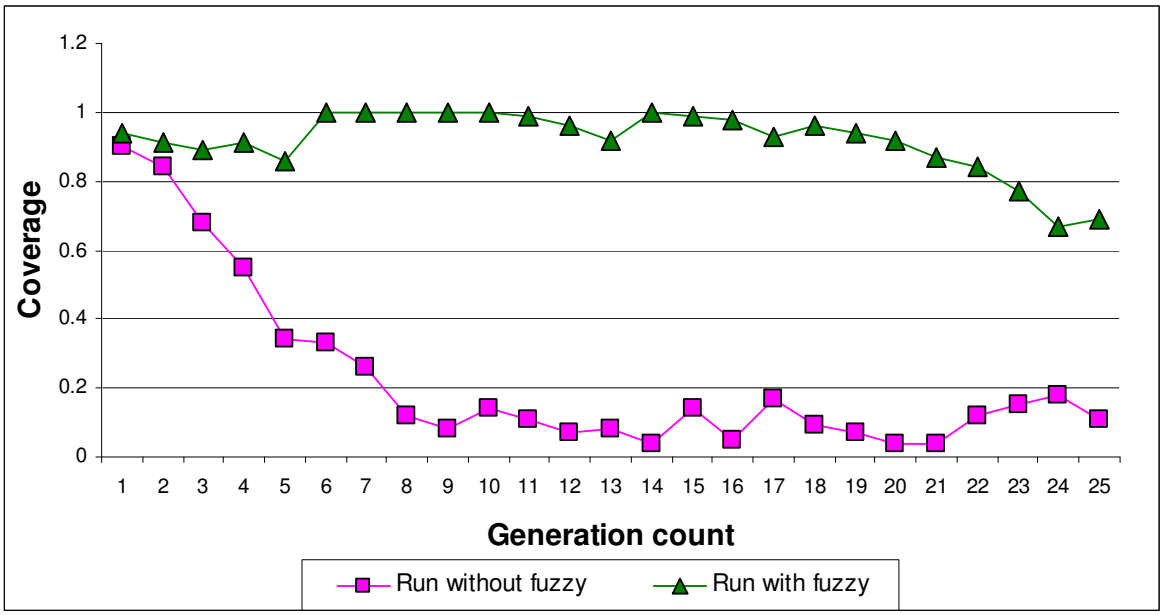


Figure 6.3-5 Coverage comparison between the run with and without fuzzy rules

The last Pareto front approximation comparison is made between the two fuzzy runs (see Figure 6.3-6). 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.

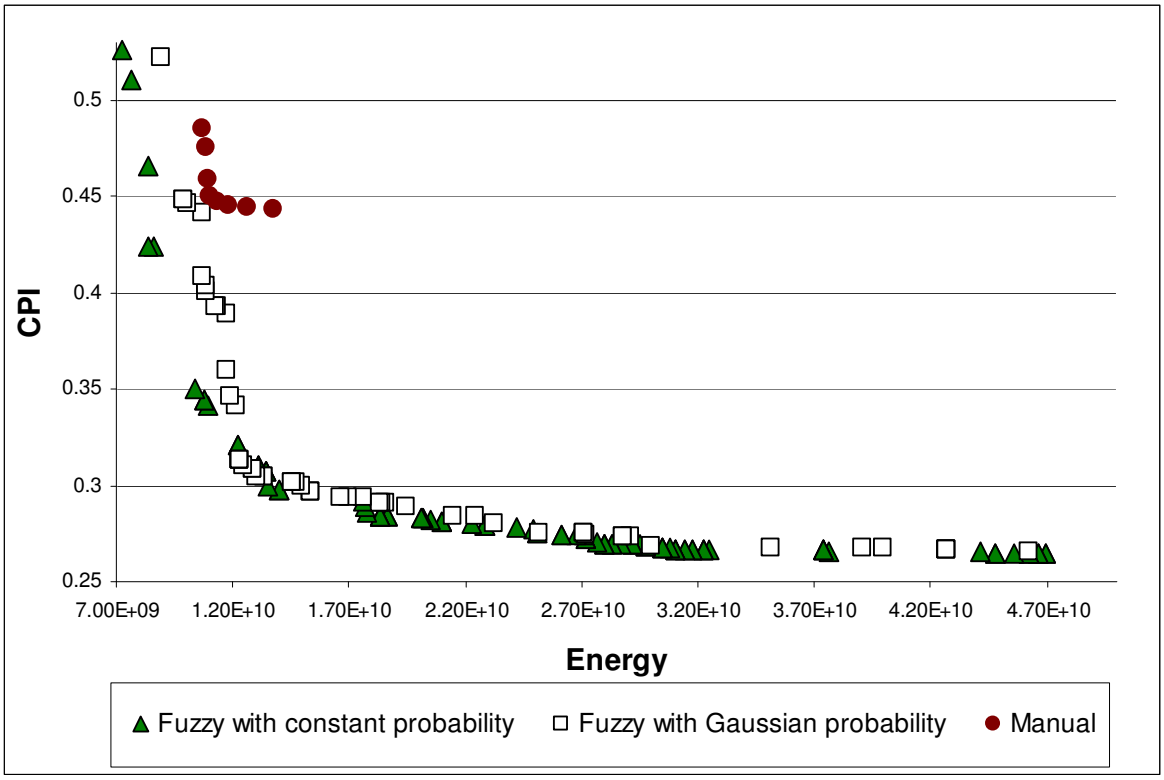


Figure 6.3-6 Pareto fronts comparison between the runs with fuzzy rules

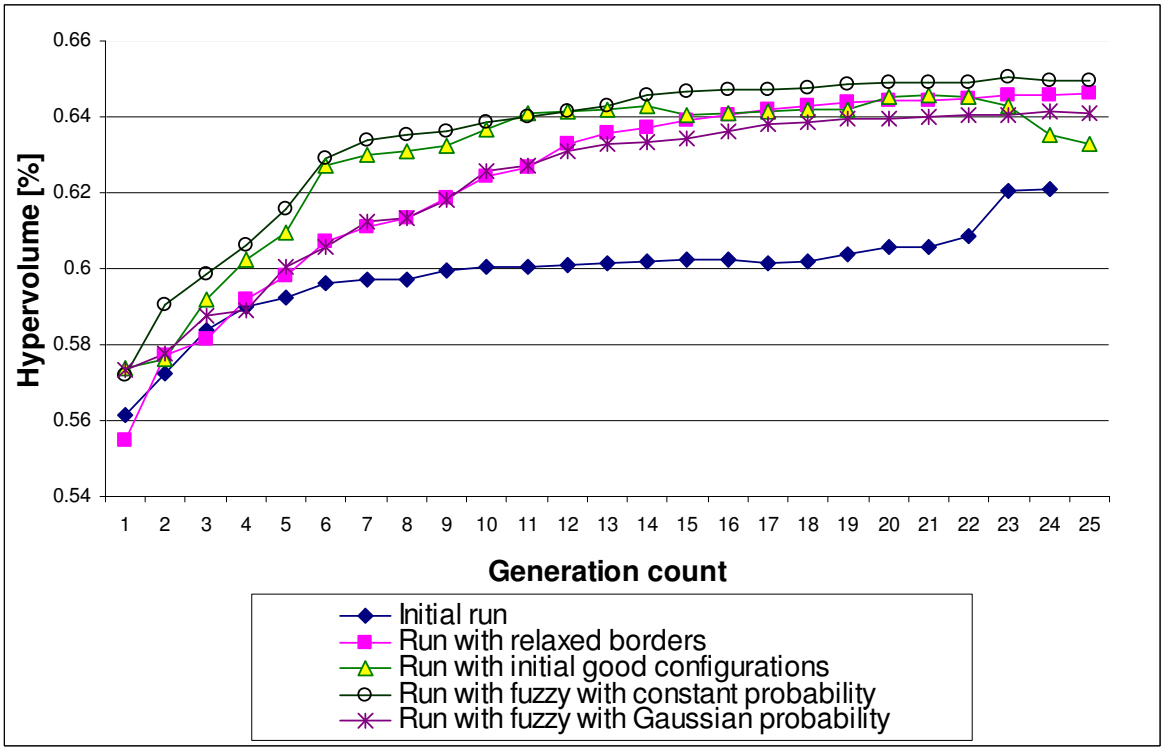


Figure 6.3-7 Hypervolume comparison

As a final comparison, we computed the hypervolume obtained by all the runs during the DSE process (see Figure 6.3-7). 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 Chapter 5.6, 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.

6.3.3 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.

6.4 Optimizing M-SIM3 Architecture

Our next objective was to move to a multi-core architecture. We analyzed many simulators (see Paragraph 6.5) 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 6.3): energy and CPI.

6.4.1 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 [90] 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 (see Paragraph. 3.6) 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.

6.4.2 Results

In our first test we ran with M-SIM 3 as single core. In Figure 6.4-1 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 6.4-2. 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).

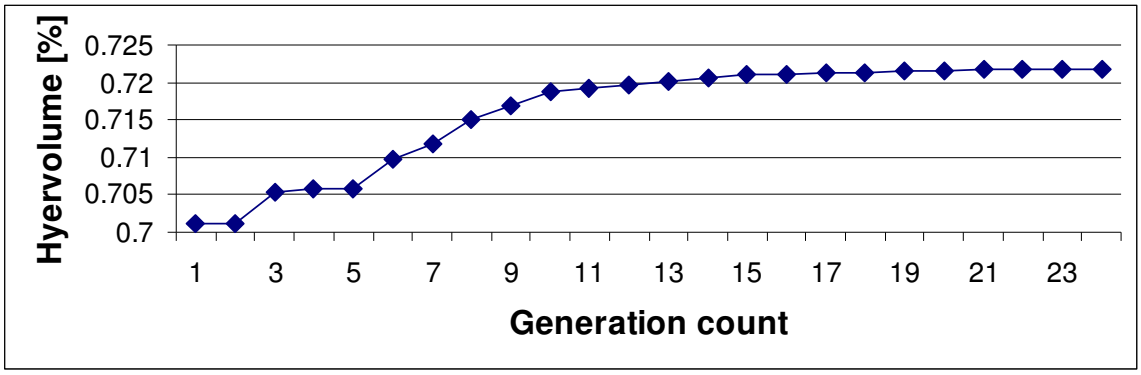


Figure 6.4-1 Hypervolume obtained for the M-SIM 3 configured as a mono-core

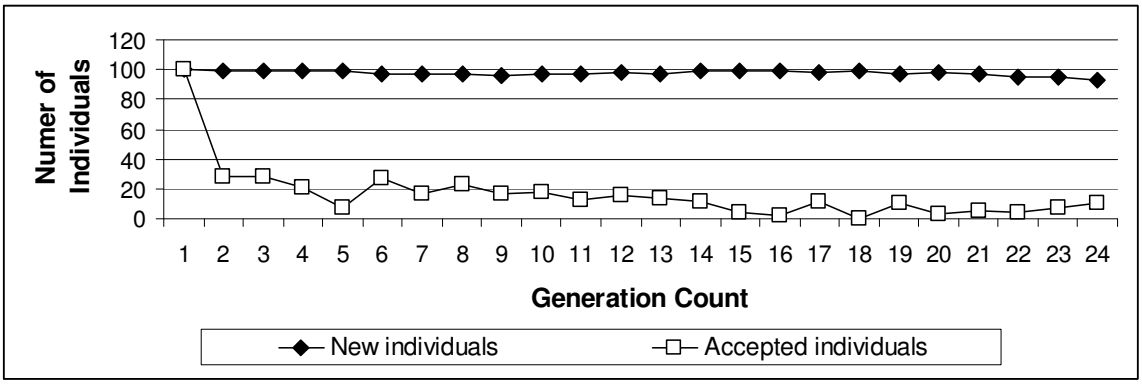


Figure 6.4-2 Number unique offspring individuals versus the number of accepted individuals in the next population – mono-core run

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 (see Paragraph 3.6.1).

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 6.4-3 shows the convergence of the algorithm. Compared to Figure 6.4-1, the evolution is more smoothly. A difference from previous exploration can be seen when comparing Figure 6.4-2 with Figure 6.4-4. In Figure 6.4-4 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 6.4-4 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).

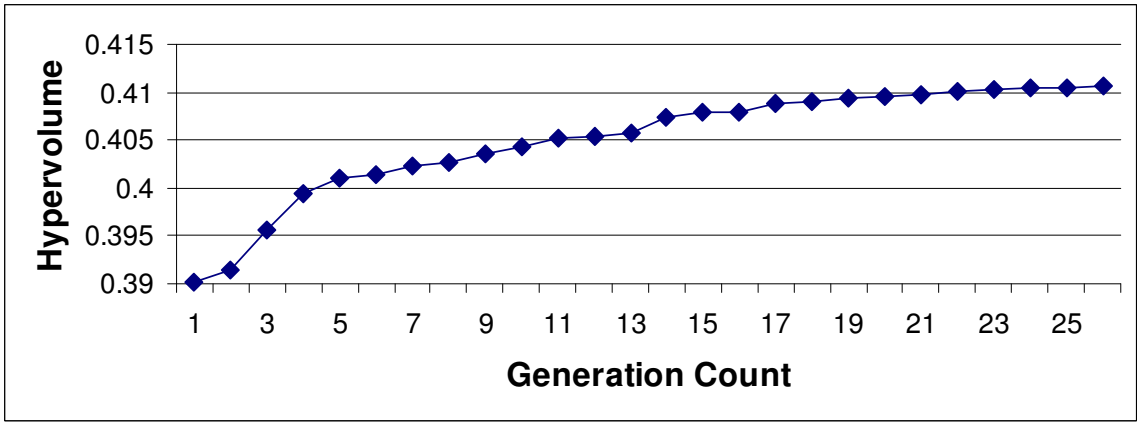


Figure 6.4-3 Hypervolume obtained for the M-SIM3 configured as a multi-core

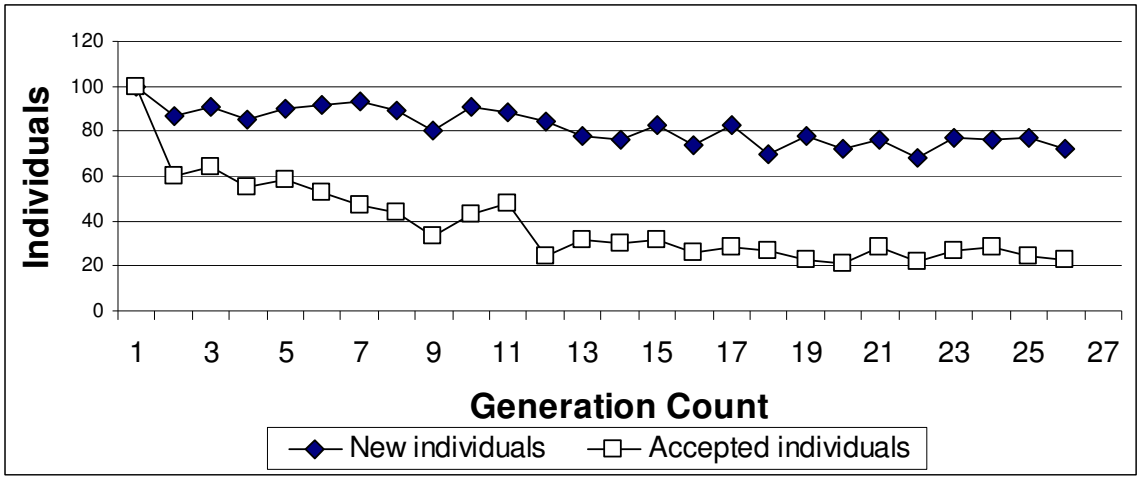


Figure 6.4-4 Number unique offspring individuals versus the number of accepted individuals in the next population – multi-core run

6.5 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.

6.5.1 UNIted SIMulation Environment

UNIted SIMulation environment (UNISIM) [105] 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 basic structure of UNISIM is shown in Figure 6.5-1.

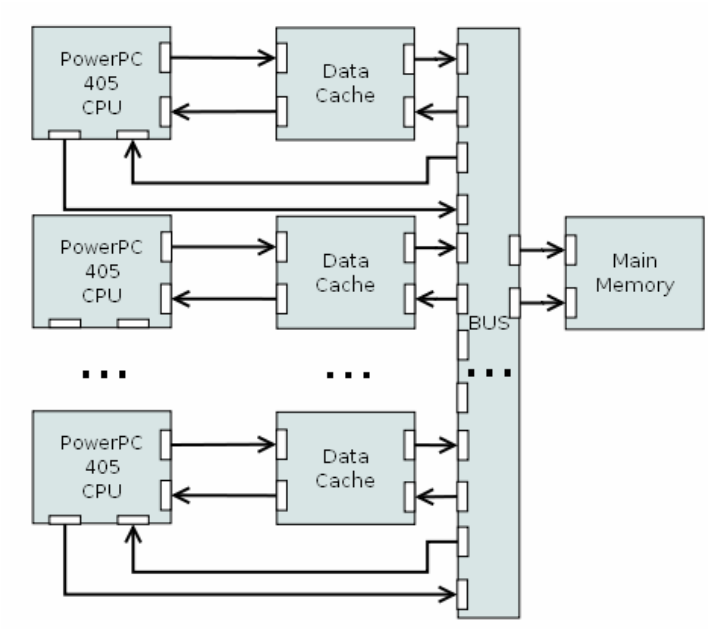


Figure 6.5-1 UNISIM cycle-by-cycle simulator structure

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 [106] and SPLASH-2 [107] 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.

6.5.2 M5

M5 [108] 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.

6.5.3 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 [84] 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.

6.5.4 SuperESCalar Simulator

SuperESCalar Simulator (SESC) [109] multi-core capable cycle level accurate simulator developed at the University of California. The advantage of this simulator is that it integrates with HotSpot [110], Wattch [94] and Cacti [111] [85]. 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.

6.5.5 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.

6.5.6 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 structure of this simulator is presented in Figure 6.5-2. 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.

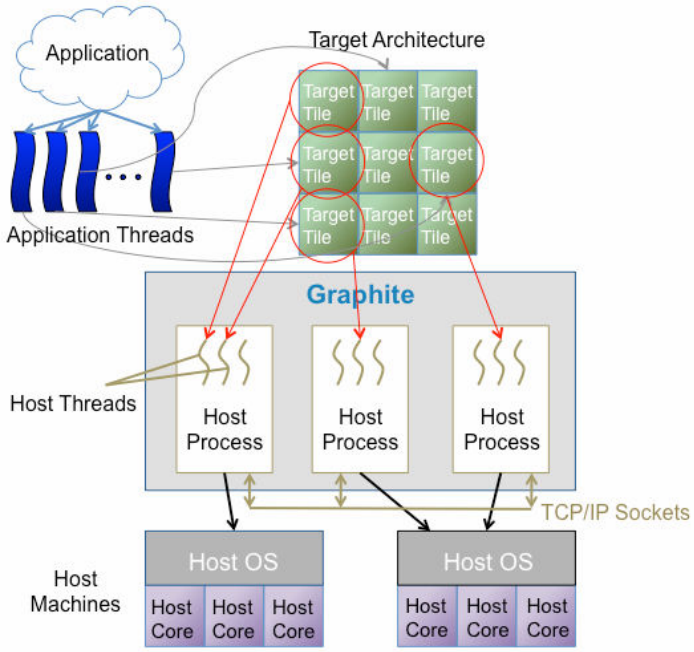


Figure 6.5-2 Graphite simulator architecture (figure taken from the Graphite web-page http://groups.csail.mit.edu/carbon/?page_id=111/)

6.6 Summary

In this chapter we focused on the M-SIM simulator, investigating both mono and multi-core architectures (M-SIM 2 and M-SIM 3). We tested some of the ideas presented in Chapter 4 (fuzzy rules, constraints, etc.) and added some new ones (initial known good configurations).

We started from a manual exploration performed by Árpád Gellért in his PhD thesis and decided to improve his results using an automatic DSE using FADSE. Since the settings differed slightly (number of simulated instructions) we redid the work performed in Gellért's thesis. We extracted the best found individuals from this experiment.

The next step was to use FADSE. The first runs were done without any prior knowledge but we did use constraints to avoid unnatural or impossible configurations. Different settings for these constraints were used. We concluded that if the good configurations are near or on the borders determined by the constraints, then the DSE algorithms are unable to find these configurations. More relaxed borders should be used even if this means increasing the design space (in our case the increase was by a factor of two).

We continued the experiments and, with these relaxed borders, we started adding knowledge to the algorithm. In the first experiment in this category we inserted some good configurations in the initial population. We used the individuals selected during the manual exploration and generated some neighbors (parameter wise). These individuals were then inserted in the initial population and the algorithm started from them. The result was a fast convergence up to a point when the algorithm stopped improving and in fact started to loose good points. We discovered that the algorithm failed in a local minimum because of the lack of diversity in the initial configuration. Still, the results were better than the ones obtained without prior information for the first generation. In conclusion, if the time available for a DSE is short, such an approach will definitely provide good results, and with some extra care in how the first individuals are selected the idea could lead to great results.

Next, we experimented with the fuzzy rules. We tried both implementations presented in Paragraph. 4.4.2.3: constant and Gaussian distribution of probability to apply the fuzzy rules. We defined a couple of rules and with them we started FADSE and let it run with the M-SIM simulator for 25 generations. The run with a constant probability provided excellent results, both in terms of convergence speed and in quality of results. None of the runs performed before provided such good results. With this we proved that using fuzzy rules given into a human readable form by an expert can lead to improvements for the DSE process. We concluded that if there are not many membership functions associated to an output variable (see Paragraph. 4.4.2) it is recommended to use the constant probability to apply the fuzzy rules.

With the M-SIM experiment, we proved that FADSE can cope with an extremely large design space ($7.7*10^{13}$) and find very good solutions by evaluating only a fraction of it (2200 configurations evaluated). This was also a reliability test for FADSE since each DSE experiment lasted for more than 3 weeks of continuous running (running with M-SIM3 took more than a month). During this time we had to restart clients and even the server (other applications running on the cluster crashed it), but FADSE recovered and continued the work and the DSE process did not have to be restarted.

We also proposed a stop criterion for the heuristic algorithms. It uses the hypervolume value evolution over the generations to see when the algorithm has converged.

Multi-core design space exploration was our next target. After analyzing several simulators like: Unisim, M5, Multi2Sim, SESC, SCoPE and Graphite we selected M-SIM3 as our simulator, mainly because of the power consumption integration. First we performed a mono-core exploration on this simulator too, but this time we let infeasible individuals in the offspring population unrestricted. We noticed a slower convergence of the algorithm, since most of the individuals (50%-60%) from the offspring population are infeasible, so the algorithm has a smaller selection pool to choose from.

Next we moved to the multi-core simulator and we showed that good results can be obtained using FADSE.

As we mentioned before, we tried using other multi-core simulators too: we tried to use and we also improved the UNISIM simulator with new cache coherency protocols, and we made the simulator easily configurable. M5 was also a choice, for which we implemented a FADSE connector. A connector was also implemented for Multi2Sim. For this simulator we also started integrating it with the McPAT power consumption estimation library. The SESC simulator was analyzed as a viable option since it has power consumption models integrated by default. Some compatibility issues between this simulator and the HPCs we had at our disposal lead to the impossibility to use this simulator. SCoPE and Graphite are two simulators which we currently try to integrate with FADSE. With SCoPE we already built some multi-core configurations.

"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 [112], 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.

7.1 Network on Chip Research Challenges

UniMap was designed to study Network-on-Chip (NoC) architectures, which basically are communication networks integrated on a single chip. These architectures represent a novel research domain (they appeared in the 90's but research only started in 2000). According to the HiPEAC's vision [113] NoC's importance will grow in the following years. Some of the papers that pioneered this research area are: Guerrier and Greiner [114], Hemani et al. [115], Dally and Towles [116], Wingard [117], Rijpkema et al. [118], Kumar et al. [119] and Micheli and Benini [120]. They were introduced to avoid global wiring structures. One of the advantages of the NoC architecture is that it allows reusability and interoperability of different Intellectual Property (IP) blocks because of the modularity and standard network interfaces. Resources like wires are shared: while one IP is not communicating another can use the wires. Packets are used for communication.

There are several problems that need to be addressed in NoC architectures [121]. In his PhD thesis [122], Ciprian Radu is addressing the Network-on-Chip application mapping problem. This problem is defined in [123] as the topological placement of IP cores onto NoC tiles. This mapping problem is an NP hard problem [124] and it is solved with heuristic algorithms. These algorithms can take into consideration different metrics like bandwidth, latency, energy consumption, routing [125], etc. Before the mapping procedure begins, the application tasks have to be assigned to IP cores. This is another problem NP hard problem, called the scheduling problem. The NoC application mapping problem also directly interacts with the routing problem, as it is shown in Figure 7.1-1.

A Communication Task Graph (CTG) [121] is a directed acyclic graph through which an application is described. The graph nodes represent application tasks. The arcs mark the communications between tasks. They are weighted with the communication volume (expressed in bits).

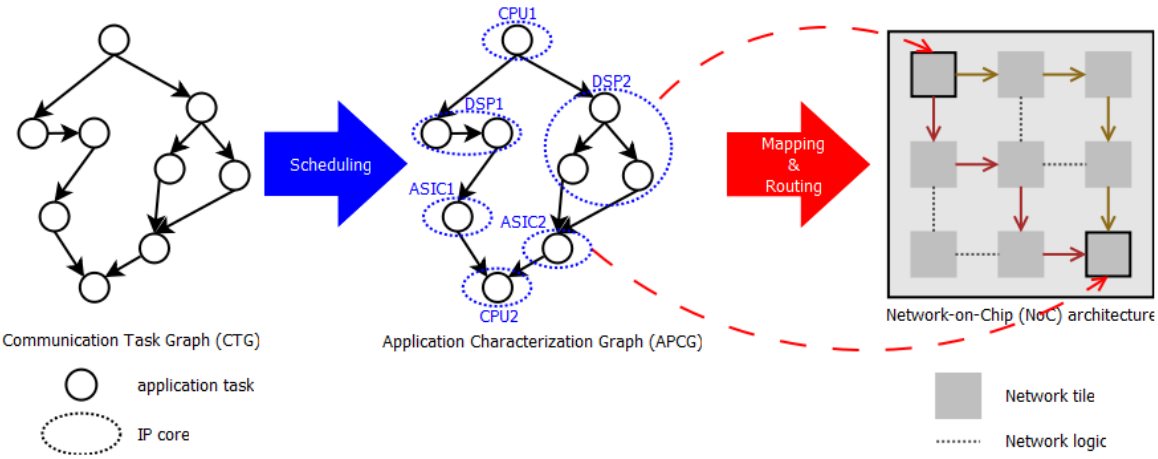


Figure 7.1-1 The scheduling, application mapping and routing problems [122]

Scheduling algorithms are employed to assign and schedule application tasks to (usually heterogeneous) IP cores. The output is a directed graph, called an Application Characterization Graph (APCG) [121].

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. The mapping algorithm must be aware of the routing algorithm, i.e. how data is sent from one network node to another.

7.2 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.

The applications and NoC architectures are described using XML based interfaces. Real applications are described only in terms of their communication patterns (using CTGs). Communication is the most important characteristic, from the NoC perspective. When the simulator is used the applications' execution is modeled using finite state machines [112].

UniMap can find near optimal mappings using specialized algorithms: simulated annealing, branch and bound [123] [125], Optimized Simulating Annealing (OSA) [126] and evolutionary algorithms [122]. The mappings can be evaluated using an analytical energy model (faster but less accurate) [123] [125] or the simulator (slower but more accurate).

The simulator included in UniMap is developed on top of the popular ns-3 simulator for Internet systems [127]. UniMap currently allows the user to specify: the packet size, packet injection rate, buffer size, network size, switching mechanism (Store-and-Forward, Virtual Cut-Through and Wormhole), routing protocol (XY, YX and two adaptive protocols that consider the network's load), network topology (2D mesh, Irvine, 2D torus, 3D mesh, 3D torus), etc. [128][112]. It also integrates the ORION 2.0 NoC power and area model [129]. CTGs are obtained either from realistic embedded applications or from the E3S benchmark suite [130] or from real-world

applications using the CETA tool [131]. UniMap's simulator provides a traffic generator based on Communication Task Graphs.

7.3 Related Work

This section presents some work related to ours, which shows the need for an application driven automatic design space exploration of Network-on-Chip architectures.

The work from [132] performs an exhaustive design space exploration of NoC designs. Network-on-Chip parameters like number of nodes, topology, routing protocol, switching mechanism, packets' and buffers' size are considered. It is concluded that, from the researched design space, *no* NoC architecture was found to be optimal for all traffic patterns. A particular network topology could for example be the most suitable one only for some traffic patterns. If there would be a topology which is best for all communication patterns, most likely network buffers will have to be allocated in a specific manner, for different applications. The authors of the mentioned paper propose a *polymorphic network*, which is a device that may be configured to emulate different Network-on-Chip designs. Before an application is run, the polymorphic network is configured like an arbitrary network. Then, the Operating System can configure the network, based on what application is executed. Network parameters like topology, buffers and link width can be set.

A polymorphic network is essentially made from many buffers and crossbars and it is obviously a flexible design. Many different networks may be created but, with the price of a high area budget.

Another example is given in [133]. Allocating input buffers for a NoC in a non-uniform manner can dramatically increase the NoC's performance due to the fact that the size of the buffers is very important when the network gets congested. The authors propose an algorithm that, based on a given traffic pattern and specific network, detects overloaded buffers and increases their capacity such that they do not become a performance bottleneck.

The above two examples suggest the very high number of possible Network-on-Chip architectures. In such a context, exhaustively searching for the best NoC design for a given application is unfeasible. Design Space Exploration techniques are required. Multi-objective heuristic algorithms can be employed to search for the near optimal System-on-Chip for a specific application, for example in terms of application runtime, energy consumption and area of integration. Because of its modularity, flexibility, ease of use and highly configurable characteristic, our developed Framework for Automatic Design Space Exploration is suitable for an automatic and application driven DSE for SoC architectures.

Using xPipesCompiler [134] one can automatically instantiate application-specific Networks-on-Chip. The building blocks of a NoC design are modeled through SystemC at cycle level. This tool has the advantage of allowing custom networks to be created. For example, for a switch that does not use its output port, the buffer and logic for that port is not generated, thus saving area and power consumption. An input file is used by xPipesCompiler to instantiate the NoC. The file contains information about and relation between IP cores, network switches and communication links. The tool does not actually work with routing algorithms. The routing paths are specified by the user, by writing them in routing tables. Another disadvantage is that the network topology is not determined based on application characteristics. It needs to be specified manually.

Next we present our approach that automatically searches for the "best" System-on-Chip design, for a given (concurrent) application. The search has three objectives: application runtime, SoC energy and SoC area. Our model integrates UniMap with FADSE. UniMap provides application mappings onto Network-on-Chip architectures. We try to find the IP cores (taken from a library of IP cores) which are the most suited for executing a particular application. We also want to determine the optimal NoC design for an application. Our three objectives are contradictory because improving one of them usually means diminishing at least one of the other two.

7.4 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. Such an approach has the following complexity: $C_{IdealDSE} = {}_n P_c \cdot N \cdot C$, where n is the number of NoC nodes, c represents the number of Intellectual Property cores, N is the number of evaluated NoC designs, for each mapping and C is the number of IP core types that can be used for the c cores executing the given application. The first factor is the number of possible mappings, which is factorial in size. $N = \prod_{k=1}^{p} P_k$, with p being the number of (ns-3) NoC parameters. P_k is the total number of values of parameter k. $C = \prod_{k=1}^{c} c_k$, with c_k being the number of IP cores capable of executing the tasks assigned to core $k = \overline{1,c}$ (our modeled System-on-Chip is heterogeneous). N and C are exponential. Therefore, it is clear that $C_{IdealDSE}$ is a huge number. Consider for example a small NoC with n = 16 nodes. There are 16! possible mappings of c = 16 Intellectual Property cores. For a Network-on-Chip with just four parameters, each with ten values, N = 10^4. Also, let us assume that there are ten IP core types capable of executing the tasks: $C = 10^{16}$. In this case $C_{IdealDSE} = 16! \cdot 10^{20} \approx 2 \cdot 10^{33}$. Considering we would need only one second per simulation, the exhaustive search would require $63 \cdot 10^{24}$ years to be completed (on a single core system). It is clear that a tool like UniMap is required to search heuristically among all possible application mappings. A Framework for Automatic Design Space Exploration is also mandatory for exploring the huge space of possible SoC architectures. Using UniMap and FADSE we limit the number of simulations. We acquire speed at the price of finding only near optimal solutions. Still, such a DSE approach is still very time demanding. For example, let us say that UniMap selects only ten thousands mappings and with FADSE we need to simulate just ten thousands SoC designs per mapping (with 1 s / simulation). In this case 300 years would still be required to finish the DSE process.

The DSE mechanism described above 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. FADSE will take the mappings from this front and search for each one the best System-on-Chip. Such a DSE workflow has the following complexity: $C_{UniMapDSE,FADSE} = {}_n P_c + P \cdot N \cdot C, \quad P << {}_n P_c$, where P is the number of UniMap DSE

Pareto mappings. Of course $C_{UniMapDSE,FADSE} < C_{IdealDSE}$ but we go farther away from the optimal solutions. For the above example, where the DSE would need 300 years, this second approach would need only 13 days. However, recall that we obtain only 13 days because we assumed only one second per simulation. This second DSE approach is also quite time consuming.

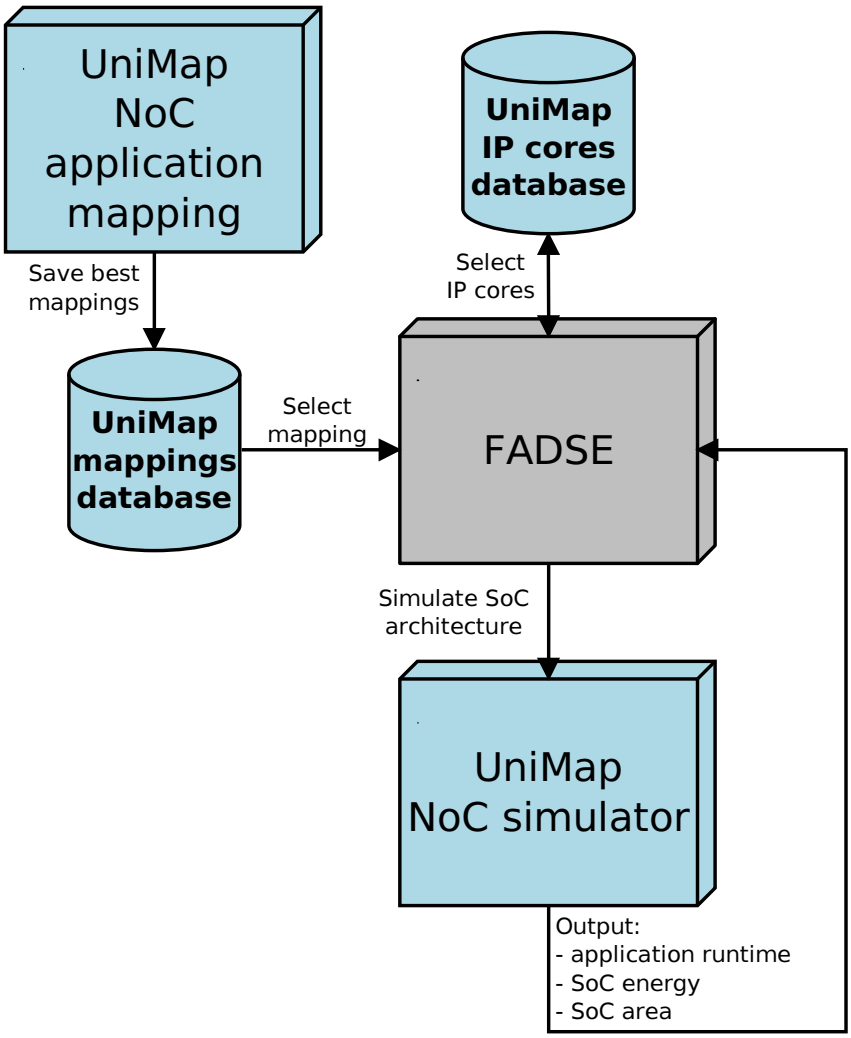


Figure 7.4-1 Application driven DSE workflow for SoC designs

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. The complexity of this DSE approach is $C_{UniMapAnalytic,FADSE} = ({}_nP_c)_{analytic} + B \cdot N \cdot C, \quad B << {}_nP_c$, where B is the number of best analytic 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 (see Chapter 3.3). A mapping was evaluated in 0.04 ms. Returning to our example, let us consider that B is ten. In such a case this DSE would need 12 days. This estimation is very close to that for the second DSE. However, in this later case the estimation is much more realistic because UniMap does not use a simulator (for which we unrealistically said it would need just a second per simulation). With this

third DSE workflow we still work under the assumption that FADSE uses a NoC simulator that needs just a second per simulation. However, we note that we can distribute simulations with FADSE on our High Performance Computing system.

We can conclude that $C_{UniMapAnalytic,FADSE} < C_{UniMapDSE,FADSE} < C_{IdealDSE}$. This is why we use in our research the third DSE approach. Figure 7.4-1 presents our automatic and application driven design space exploration workflow for System-on-Chip designs.

Our Design Space Exploration workflow begins from UniMap, which maps applications onto Network-on-Chip designs. Each mapping is evaluated with an analytical model [135] 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 considered by FADSE (for each application) for architectural optimization. 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 [122].

Afterwards 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 informs about 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. Next, 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 [130] 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 [129]. 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 exactly how 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 and modifying it would

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

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, too. In order to do so, we would need to adapt our mapping algorithms for these other Network-on-Chip topologies as well.

7.5 Methodology

Since we present in this thesis only preliminary results, we worked 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 chose 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.

7.6 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 on the *telecom* benchmark. On the other benchmarks we were able to explore a single mapping: the best one found analytically.

7.6.1 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 7.6-1. 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.

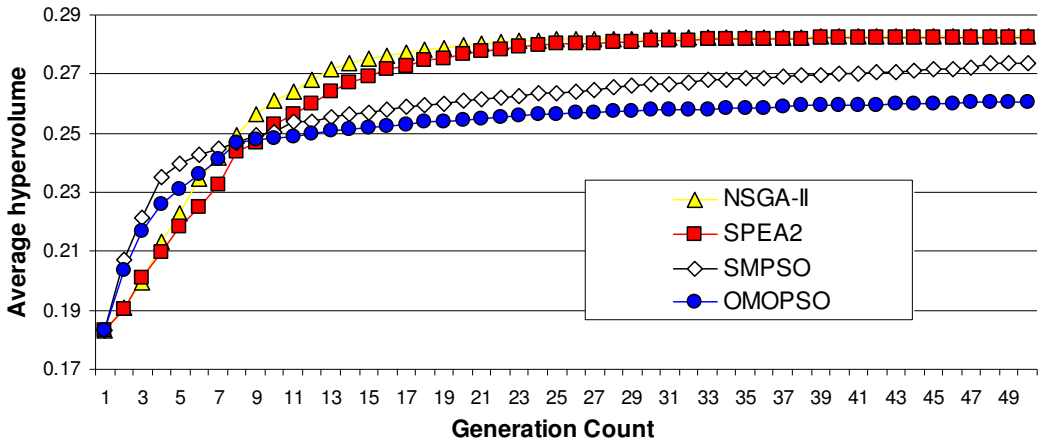


Figure 7.6-1 Average hypervolume over all ten best *telecom* mappings

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 7.6-2.

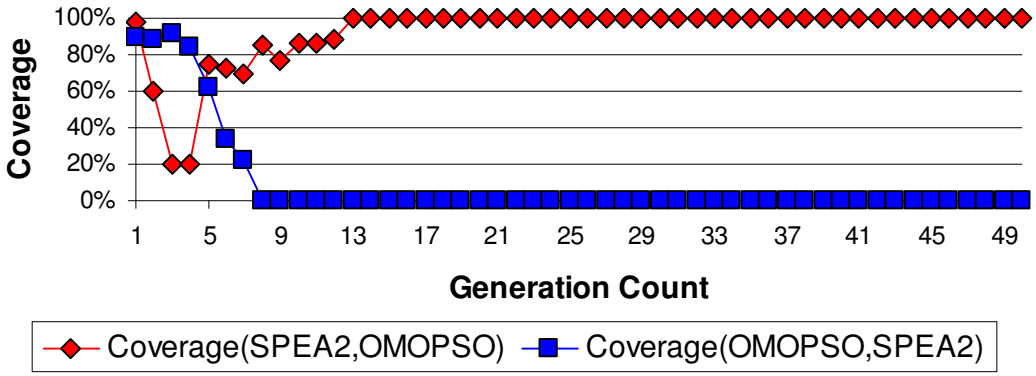


Figure 7.6-2 Coverage comparison between SPEA2 and SMPSO on the *telecom* benchmark

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 7.6-3. 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.

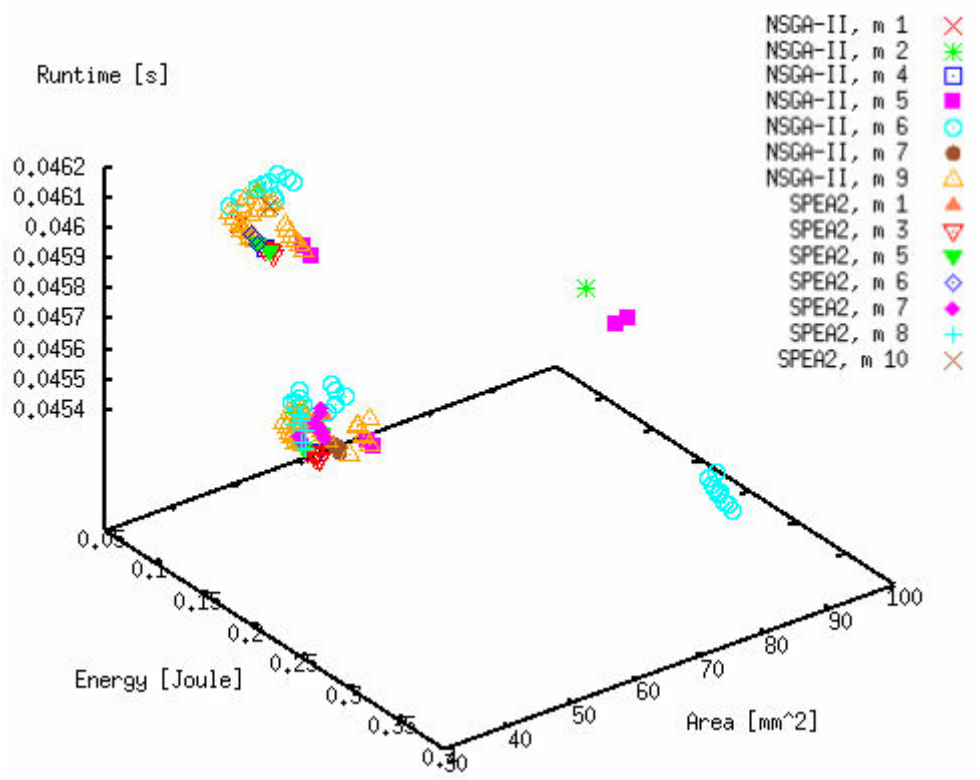


Figure 7.6-3 Best configurations found by all four algorithms for *telecom* benchmark

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 [122].

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.

7.6.2 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 7.6-4 we present the hypervolume value obtained by each algorithm during the 50 generations.

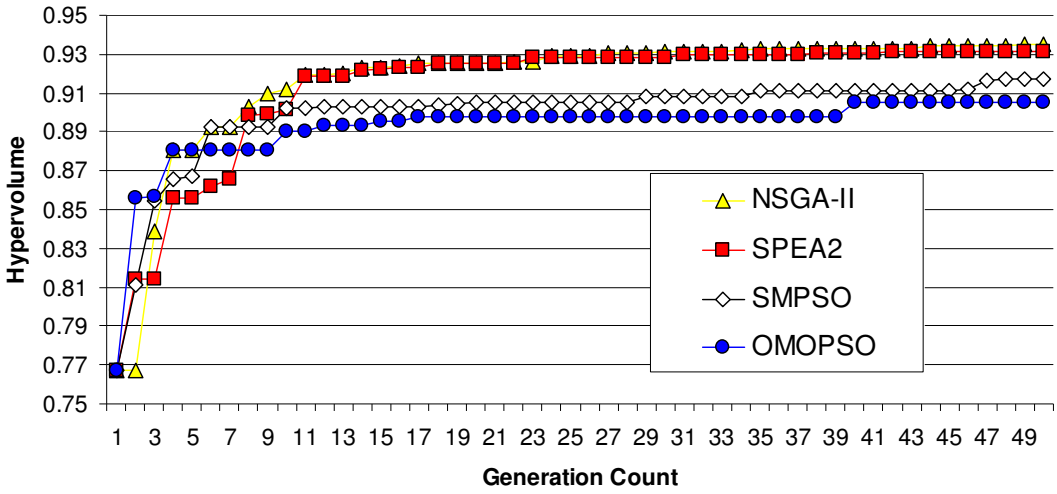


Figure 7.6-4 Hypervolume obtained by all the algorithms for the *MPEG-4* benchmark

Since we ran only on one mapping and we do not present an average result, 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 perspective it seems that NSGA-II finds the best results. From the point of view 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 7.6-5. 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.

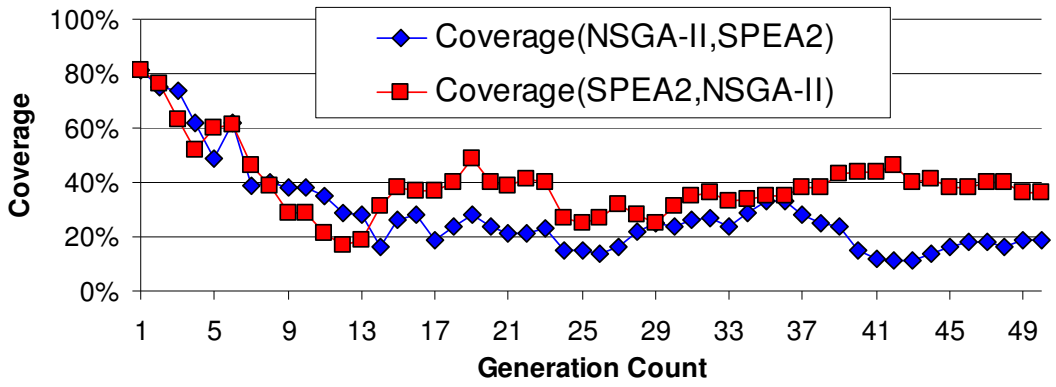


Figure 7.6-5 Coverage comparison between NSGA-II and SPEA2, for *MPEG-4*

Second, in Figure 7.6-6 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 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 7.6-7. 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.

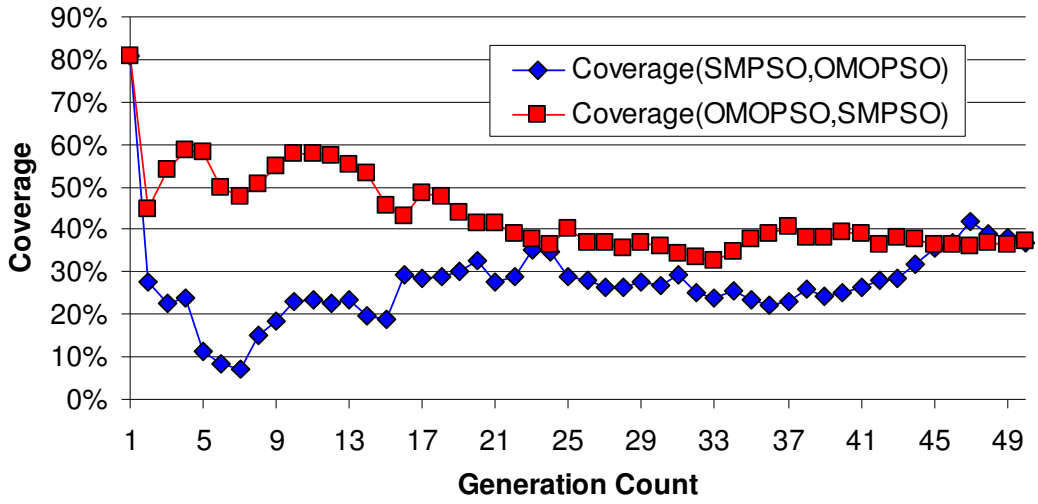


Figure 7.6-6 Coverage comparison between SMPSO and OMOPSO, for *MPEG-4*

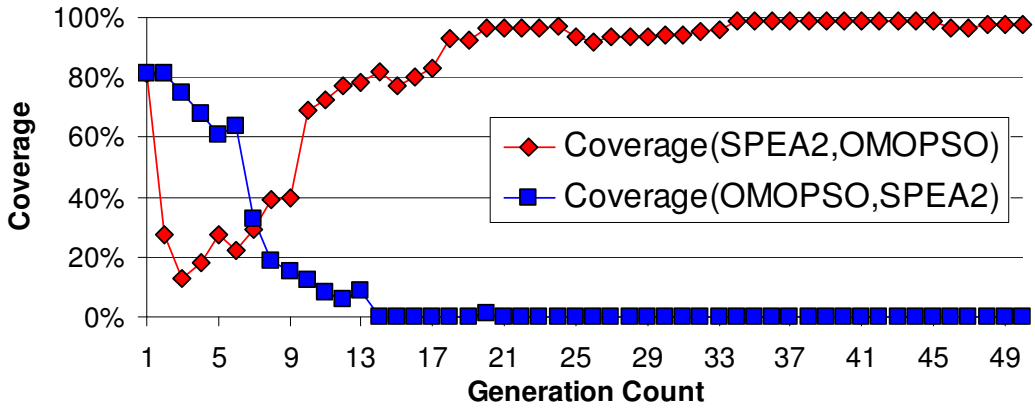


Figure 7.6-7 Coverage comparison between SPEA2 and OMOPSO, for *MPEG-4*

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 7.6-8 (since this is a joint work, we presented this figure also in [122]). 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).

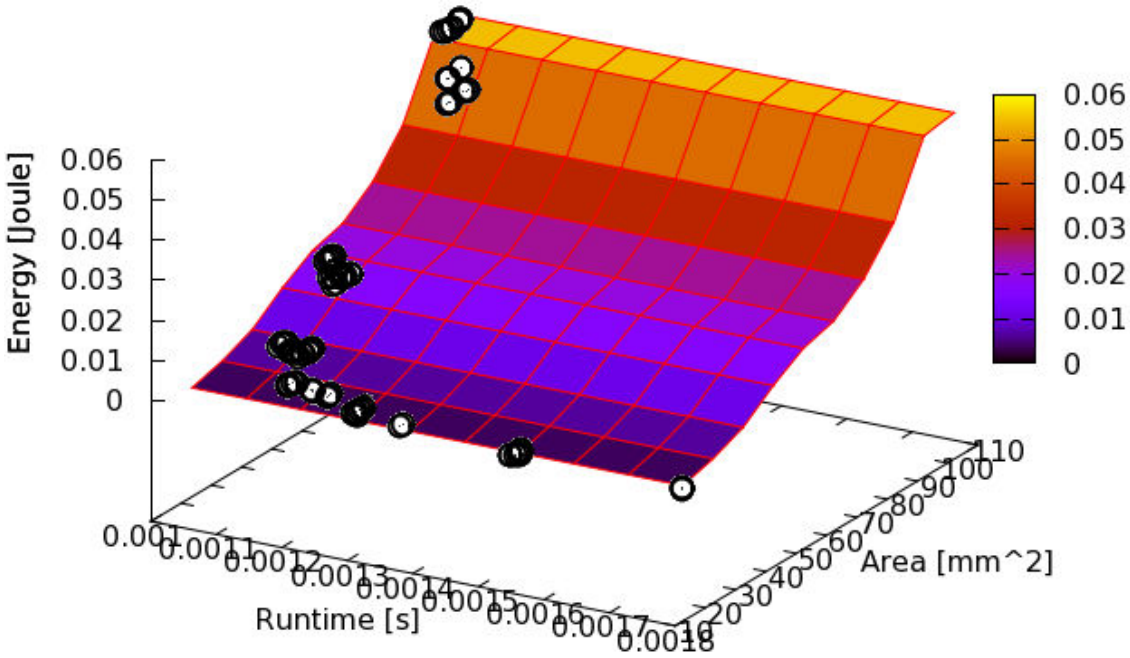


Figure 7.6-8 Surface obtained by interpolating the points in the objective space found by NSGA-II

7.6.3 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 7.6-9 and Figure 7.6-10.

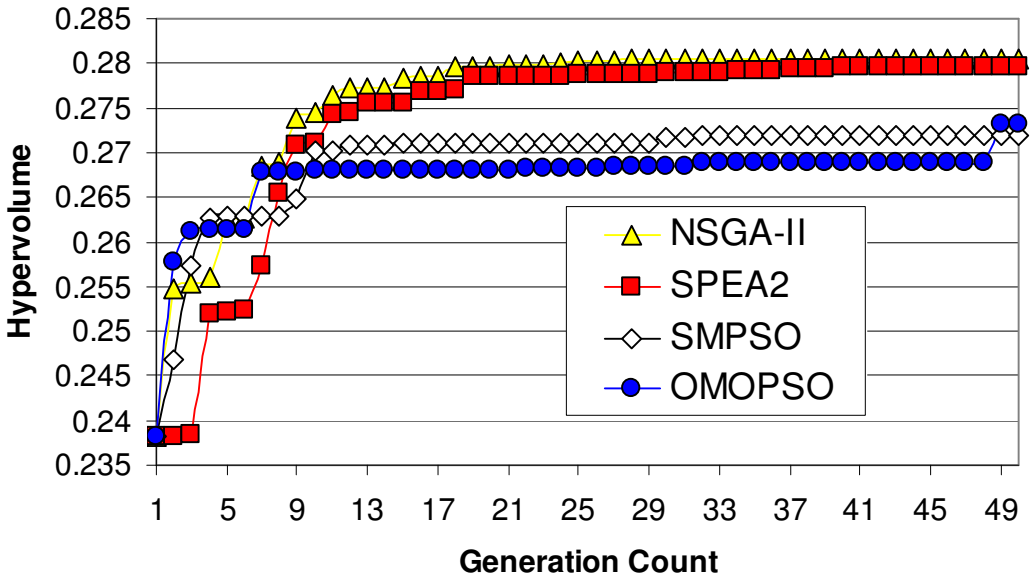


Figure 7.6-9 Hypervolume obtained by all the algorithms for the *H.264* benchmark

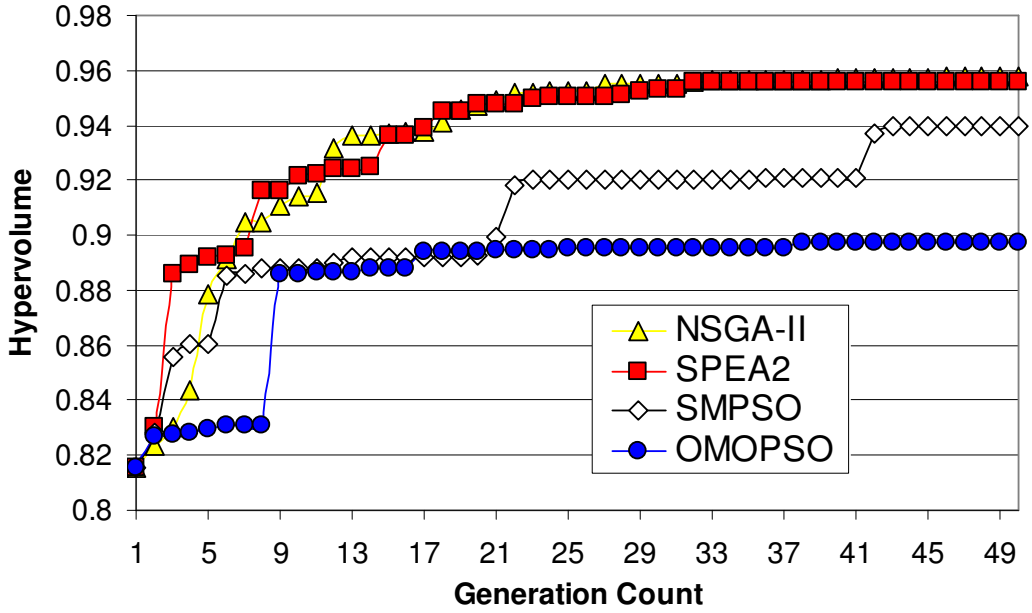


Figure 7.6-10 Hypervolume obtained by all the benchmarks for the *VOPD* benchmark

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.

7.7 Improving the MANJAC Many-core System

The MANJAC [136] is a system with 64 native JAVA execution multi-core, multi-threaded processors arranged into an 8x8 mesh. Each processor (called Jamuth [137]) contains 6 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 [138].

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 suggested 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" [138].

7.8 Summary

In this chapter we performed automatic designs pace exploration of SoC architectures. We focused also on the application mapping problem and we proposed a DSE process which includes this aspect too. Basically, our idea was to serialize the application mapping problem and the DSE of the SoC architecture. We start by finding analytically a set of best mappings. These mappings are then given to FADSE which tries to find good parameters of the SoC architecture. Besides NoC parameters like operating frequency, buffer size, etc., FADSE also chooses the cores for each task from an IP library.

To perform this DSE we integrated FADSE with the Unified framework for Network-on-Chip application Mapping (UniMap) which is responsible (in this experiment) with finding the best mapping analytically. UniMap integrates as well the simulator FADSE uses to obtain the objectives to optimize.

With this experiment we tested FADSE with a problem with three objectives: energy, area and speed (application runtime). We can conclude that, using the algorithms integrated in FADSE, the user can optimize problems with many objectives.

An interesting conclusion that we can draw, which is different from what we have seen in previous chapters, is that the genetic algorithms perform much better than the PSO algorithms in terms of quality of results. All the best configurations are found with the genetic algorithms (NSGA-II and SPEA2) and none with OMOPSO and/or SMPSO. It is interesting that the results seem to depend more on the algorithm type than on the specific implementation. Still, we observed a faster convergence speed of the PSO algorithms in most of our tests, but the genetic algorithms quickly overcome this drawback.

We want to emphasize that this is still a preliminary research. We want to include domain knowledge and make use of all the features available in FADSE to improve the quality of results and convergence speed of the algorithms.

At the end of this chapter we present some work done during the 5 months mobility period at the Augsburg University in Germany.

"The outcome of any serious research can only be to make two questions grow where only one grew before."

Thorstein Veblen

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 lead 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 Glossary

Architecture	The structure of a computer system
Bandwidth	It is the measure of available/consumed data by communication resources. It is measured in bits/second or multiples of it (kilobits/s, megabits/s etc).
Benchmark	It is a program used to evaluate computer architectures.
Centroid	The point with minimal average distance to all other points in the set.
Checkpointing	A method that saves the current state of the program so it can be restarted in case of failure from that point as nothing happened.
Crossover	Is a genetic operator which combines two individuals (parents) to obtain two new individuals (offspring) that inherit characteristics of the parents
Diversity	It expresses the uniqueness of individuals from a population. It can also be seen as a measure of average distances between individuals. If it is high then the population is diverse.
DSE	It is the process of searching through the possible hardware and/or software configurations to find optimal design points.
Elitism/ elitist selection	It is a property of selection methods that guarantee that the best individuals survive in the next generation.
Evolutionary algorithm (EA)	A term that describe all the stochastic heuristic algorithms inspired by Darwinian evolution.
Fitness	It is a number associated to an individual that influences the probability for reproduction (probability of being selected as parent). It is usually based on the objectives value.
Gene	Is the subunit of a chromosome, it represents a parameter.
Generation	It is the iteration of an evolutionary algorithm. It includes the selection, of parents, creation of offspring, evaluation, creation of the next parent population.
Genetic algorithm (GA)	Algorithms inspired by Darwin's theory of evolution. The offspring are produced through crossover and mutation.
Greedy search	A search method that always makes the locally optimal choice with the hope of finding the global optimum.
Heterogeneous multi-core systems	A system that incorporates a variety of different types of processing units (general purpose processors, special purpose processors, graphic processing unit, etc.).
Hill climbing method	It is a search method. It usually starts from a random solution. Generates neighbors of this solution and if it founds a better one moves to this new point. The algorithm is repeated until no better solution is found. It can easily fall into local minima.
Individual	Is the unit of selection, it caries the genetic information (chromosome) but also contains information about the values of the objectives.
Instructions per clock cycle (IPC)	It measures how many instructions a processor can execute in parallel, so it is a measure of concurrency. It is used to measure the performance of computer processors.

Latency	It measures the delay experienced in a system.
Local minima/maxima	The minimum/maximum of a function within a neighborhood. It can be the global minimum/maximum, but this is not a requirement.
Multi-objective optimization	Optimization with the purpose is to improve multiple objective functions simultaneously. The objectives are usually conflicting so no "best" individual can be found.
Mutation/ mutation operator	An operator used mainly by genetic algorithms which changes a parameter (gene) randomly with a certain probability.
NP-hard (Non-Deterministic Polynomial-hard)	A NP-hard problem is a problem at least as hard as a NP-complete problem. In an NP-complete problem the complexity increases exponentially with the number of input parameters. Using an enumerative search strategy requires an exponential increasing time to solve it.
Offspring	The individuals obtained after reproduction in an evolutionary algorithm, the children.
Parent	An individual that was selected for reproduction.
Pareto optimality	Optimality criterion used in multi-objective problems. A point A is said to be Pareto optimal if there is no other point B dominating point A (better on all the objectives).
Pareto set/ Pareto front	Set in parameter space, front in the objective space, is the group of individuals that are Pareto optimal.
Particle swarm optimization (PSO)	It is an optimization method inspired by the way birds search for food. It uses the terms of swarm instead of population and particle for individual. The particles are changed according to: global information (the current leader of the swarm, the best performing particle) and local information (the best position the current particle had in its entire existence).
Power consumption	The dynamic power consumption is defined as: $P_d = C \cdot V_{dd}^2 \cdot a \cdot f$, where C is the capacitance, V_{dd} is the supply voltage, f is the clock frequency and a is an average number for how often clock ticks lead to switching activity.
Search space	All the possible configurations that can be obtained with a set of parameters.
Selection/ selection operator	Operator used by evolutionary algorithms which chooses the individuals from the current population that will become parents.
Simulated annealing	It is a generic search method inspired from the annealing process from metallurgy. It is similar with hill climbing, the difference being the fact that it allows the algorithm to choose worse individuals as the current solution. The probability of choosing a worse value decreases over time in concordance to a variable called: temperature.
Simulator	A software program that simulates a computer system with the purpose of evaluating the simulated architecture.
Simultaneous multithreading (SMT)	The architectures that, in order to better reutilize the resources available in superscalar processors, allow multiple threads to execute in parallel on some structures.

Stochastic	Means random. A stochastic algorithm behavior is non-deterministic. The next state will be dependent on both the deterministic predictable actions of the algorithm and also by a random element.
Subpopulation	Set of individuals that are isolated from the rest of the population. Only individuals from the same subpopulation can produce offspring. Usually there is some sort of communication between populations at certain times in the algorithm, when individuals might migrate from one subpopulation to another.
Superscalar processor	A microarchitecture that allows multiple instructions being executed in parallel. This is done using hardware resources multiplication and multiple pipelines.
Threshold	A value above which something is true or something happens and below which it is not true or does not happen.
Topology	The layout pattern of interconnections of the various elements (links, nodes, etc.) of a computer network.
Tournament selection	A method of selecting an individual from a population of individuals in a genetic algorithm. Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover. Selection pressure is easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

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

[7] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons, 2001.

[8] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 1995, vol. 4, pp. 1942-1948 vol.4.

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

[10] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Systems, Man, and Cybernetics, 1997. "Computational Cybernetics and Simulation"., 1997 IEEE International Conference on*, 1997, vol. 5, pp. 4108 vol.5, 4104.

[11] 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.

[12] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, p. 221--248, 1994.

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

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

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

[16] B. W. Silverman, *Density estimation for statistics and data analysis*. CRC Press, 1986.

[17] 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.

[18] ***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.

[19] 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.

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

[21] 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.

[22] J. Durillo, J. García-Nieto, A. Nebro, C. Coello, F. Luna, and E. Alba, "Multi-objective particle swarm optimizers: An experimental comparison," in *Evolutionary Multi-Criterion Optimization*, 2009, pp. 495–509.

[23] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary computation*, vol. 8, no. 2, p. 195, 2000.

[24] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[25] T. Matsui, K. Kato, M. Sakawa, T. Uno, and K. Matsumoto, "Particle Swarm Optimization for Nonlinear Integer Programming Problems," *Proceedings of International MultiConference of Engineers and Computer Scientists 2008*, pp. 1874–1877, 2008.

[26] E. C. Laskari, K. E. Parsopoulos, and M. N. Vrahatis, "Particle swarm optimization for integer programming," in *Computational Intelligence, Proceedings of the World on Congress on*, Los Alamitos, CA, USA, 2002, vol. 2, pp. 1582-1587.

[27] 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.

[28] P. Czyzak and A. Jaszkiewicz, "Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization," *Journal of Multi-Criteria Decision Analysis*, vol. 7, no. 1, pp. 47, 34, 1998.

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

[30] ***H. Calborean***, "An overview of the multiobjective optimization methods," Computer Science Department, "Lucian Blaga" University of Sibiu, Sibiu, Romania, 2, 2010.

[31] 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.

[32] 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.

[33] J. Knowles and D. Corne, “The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation,” in *Proceedings of the congress on evolutionary computation*, 1999, vol. 1, pp. 98–105.
[34] D. W. Corne, N. R. Jerram, J. D. Knowles, M. J. Oates, and M. J, “PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization,” *Proceedings Of The Genetic And Evolutionary Computation Conference (Gecco)*, p. 283--290, 2001.
[35] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, “MOCell: A cellular genetic algorithm for multiobjective optimization,” *International Journal of Intelligent Systems*, vol. 24, no. 7, pp. 726–746, 2009.
[36] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, “Design Issues in a Multiobjective Cellular Genetic Algorithm,” in *Evolutionary Multi-Criterion Optimization. 4th International Conference, EMO 2007*, 2007, vol. 4403, pp. 126-140.
[37] A. J. Nebro, F. Luna, E. Alba, A. B. B, and B. Dorronsoro, “AbYSS: Adapting Scatter Search for Multiobjective Optimization,” *IEEE Transactions on Evolutionary Computation*, 2006.
[38] H. Li and Q. Zhang, “Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II,” *IEEE Trans. on Evolutionary Computation*, 2008.
[39] S. Kukkonen and J. Lampinen, “GDE3: The third evolution step of generalized differential evolution,” in *The 2005 IEEE Congress on Evolutionary Computation, 2005*, 2005, pp. 443–450.
[40] H. Eskandari, C. Geiger, and G. Lamont, “FastPGA: A dynamic population sizing approach for solving expensive multiobjective optimization problems,” in *Evolutionary Multi-Criterion Optimization*, 2007, pp. 141–155.
[41] E. Zitzler and S. Künzli, “Indicator-based selection in multiobjective search,” in *Parallel Problem Solving from Nature-PPSN VIII*, 2004, pp. 832–842.
[42] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, “Scalable test problems for evolutionary multiobjective optimization,” *Evolutionary Multiobjective Optimization*, pp. 105–145, 2005.
[43] F. Kursawe, “A variant of evolution strategies for vector optimization,” *Parallel Problem Solving from Nature*, pp. 193–197, 1991.
[44] C. M. Fonseca and P. J. Fleming, “Multiobjective optimization and multiple constraint handling with evolutionary algorithms. II. Application example,” *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol. 28, no. 1, pp. 38–47, 1998.
[45] 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.
[46] 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.
[47] V. Desmet, S. Girbal, O. Temam, and B. F. France, “Archexplorer. org: Joint compiler/hardware exploration for fair comparison of architectures,” in *INTERACT workshop at HPCA*, 2009.
[48] Z. J. Jia, A. D. Pimentel, M. Thompson, T. Bautista, and A. Nú\ nez, “Nasa: A generic infrastructure for system-level MP-SoC design space exploration,” in

Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on, 2010, pp. 41–50.

[49] S. Kang and R. Kumar, "Magellan: a search and machine learning-based framework for fast multi-core design space exploration and optimization," in *Proceedings of the conference on Design, automation and test in Europe*, Munich, Germany, 2008, pp. 1432-1437.

[50] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An integrated electronic system design environment," *Computer*, vol. 36, no. 4, pp. 45–52, 2003.

[51] A. Bakshi, V. K. Prasanna, and A. Ledeczi, "MILAN: A model based integrated simulation framework for design of embedded systems," in *ACM Sigplan Notices*, 2001, vol. 36, pp. 82–93.

[52] L. Thiele, S. Chakraborty, M. Gries, and S. K\ünzli, "A framework for evaluating design tradeoffs in packet processing architectures," in *Proceedings of the 39th annual Design Automation Conference*, 2002, pp. 880–885.

[53] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés, "FAMA: Tooling a framework for the automated analysis of feature models," in *Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems*, 2007.

[54] M. Mendonca, M. Branco, and D. Cowan, "SPLOT: software product lines online tools," in *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, 2009, pp. 761–762.

[55] ***H. Calborean***, "Developing a framework for ADSE which connects to multicore simulators," Computer Science Department, "Lucian Blaga" University of Sibiu, Sibiu, Romania, 1.

[56] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti, "Efficient design space exploration for application specific systems-on-a-chip," *Journal of Systems Architecture*, vol. 53, no. 10, pp. 733-750, Oct. 2007.

[57] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, "An efficient design space exploration methodology for multicluster vliw architectures based on artificial neural networks," in *Proc. IFIP International Conference on Very Large Scale Integration VLSI-SoC 2008*, Rhodes Islands, Grece, 2008, pp. 213-218.

[58] L. A. Zadeh, "Fuzzy sets*," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[59] J. Jantzen, "Tutorial on fuzzy logic," *Curso on-line de Fuzzy*, 1998.

[60] G. Chen and T. T. Pham, *Introduction to Fuzzy Sets, Fuzzy Logic, and Fuzzy Control Systems*, 1st ed. CRC Press, 2000.

[61] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 3, pp. 28–44, 1973.

[62] J. B. Kiszka, M. E. Kochanska, and D. S. Sliwinska, "The influence of some fuzzy implication operators on the accuracy of a fuzzy model-Part I," *Fuzzy Sets and Systems*, vol. 15, no. 2, pp. 111-128, Mar. 1985.

[63] J. Lukasiewicz, *Selected works (Studies in logic and the foundations of mathematics)*, 1st ed. North-Holland Pub. Co, 1970.

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

[65] T. Takagi and M. Sugeno, “Fuzzy Identification of Systems and Its Applications to Modeling and Control,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, no. 1, pp. 116–132, Feb. 1985.
[66] 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.
[67] J.-S. R. Jang, C.-T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, 1st ed. Prentice Hall, 1997.
[68] 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.
[69] T. Ross, *Fuzzy Logic with Engineering Applications*, 2nd ed. Wiley, 2004.
[70] “IEC 61131-7 standard.” [Online]. Available: http://www.fuzzytech.com/binaries/ieccd1.pdf.
[71] “International Electrotechnical Commission.” [Online]. Available: http://www.iec.ch/.
[72] P. Cingolani, “jFuzzyLogic.” [Online]. Available: http://jfuzzylogic.sourceforge.net/html/index.html.
[73] ***H. Calborean***, “An overview of the features implemented in FADSE,” Computer Science Department, “Lucian Blaga” University of Sibiu, Sibiu, Romania, 3, 2011.
[74] 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.
[75] ***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.
[76] 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.
[77] 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.
[78] 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.
[79] 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.
[80] 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.

[81] 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.

[82] W. Huang, M. R. Stan, and K. Skadron, "Parameterized physical compact thermal modeling," *Components and Packaging Technologies, IEEE Transactions on*, vol. 28, no. 4, pp. 615 – 622, Dec. 2005.

[83] S. Gupta, S. W. Keckler, and D. Burger, "Technology independent area and delay estimates for microprocessor building blocks," *University of Texas at Austin Technical Report TR2000-05*, 2000.

[84] 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.

[85] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," *HP Laboratories, April*, vol. 2, 2008.

[86] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *wwc*, 2001, pp. 3–14.

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

[88] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] 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.

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

[96] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," *ACM SIGOPS Operating Systems Review*, vol. 30, no. 5, pp. 138–147, 1996.

[97] O. Mutlu, H. Kim, and Y. N. Patt, "Address-value delta (AVD) prediction: A hardware technique for efficiently parallelizing dependent cache misses," *IEEE Transactions on Computers*, pp. 1491–1508, 2006.

[98] C. H. Liao and J. J. Shieh, "Exploiting speculative value reuse using value prediction," in *Australian Computer Science Communications*, Melbourne, Australia, 2002, vol. 24, pp. 101–108.

[99] S. C. Chang, W. Y. . Li, Y. J. Kuo, and C. P. Chung, "Early load: hiding load latency in deep pipeline processor," in *Computer Systems Architecture Conference, 2008. ACSAC 2008. 13th Asia-Pacific*, Taiwan, 2008, pp. 1–8.

[100] T. Sato and I. Arita, "Reducing energy consumption via low-cost value prediction," *Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*, pp. 123–137, 2002.

[101] R. Bhargava and L. K. John, "Latency and energy aware value prediction for high-frequency processors," in *Proceedings of the 16th international conference on Supercomputing*, New York, USA, 2002, pp. 45–56.

[102] L. N. Vintan, A. Florea, and A. Gellert, "Focalising dynamic value prediction to CPU's context," in *Computers and Digital Techniques, IEEE Proceedings*, Stevenage, United Kingdom, 2005, vol. 152, pp. 473–481.

[103] B. Calder, G. Reinman, and D. M. Tullsen, "Selective value prediction," in *Proceedings of the 26th annual international symposium on Computer architecture*, Washington, DC, USA, 1999, pp. 64–74.

[104] 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).*

[105] 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.

[106] C. Bienia, "Benchmarking Modern Multiprocessors," Princeton University, 2011.

[107] 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.

[108] 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.

[109] J. Renau et al., *SESC simulator*. 2005.

[110] 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.

[111] 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.

[112] C. Radu and L. Vinţan, "UNIMAP: UNIFIED FRAMEWORK FOR NETWORK-ON-CHIP APPLICATION MAPPING RESEARCH," *Acta Universitatis Cibiniensis â€“ Technical Series*, May 2011.

[113] M. Duranton et al., “The HiPEAC Vision,” *HiPEAC Roadmap*, 2010. [Online]. Available: http://www.hipeac.net/system/files/LR_3910_hipeac_roadmap-2010-v3.pdf.
[114] P. Guerrier and A. Greiner, “A generic architecture for on-chip packet-switched interconnections,” *Proceedings of the conference on Design, automation and test in Europe*, pp. 250–256, 2000.
[115] A. Hemani et al., “Network on chip: An architecture for billion transistor era,” in *Proceeding of the IEEE NorChip Conference*, 2000, pp. 166–173.
[116] W. J. Dally and B. Towles, “Route packets, not wires: on-chip inteconnection networks,” in *Proceedings of the 38th annual Design Automation Conference*, Las Vegas, Nevada, United States, 2001, pp. 684-689.
[117] D. Wingard, “Micronetwork-based integration for SOCs: 673,” *Proceedings of the 38th annual Design Automation Conference*, p. 677–, 2001.
[118] E. Rijpkema, K. Goossens, and P. Wielage, “A Router Architecture for Networks on Silicon,” *In Proceedings of Progress 2001, 2nd Workshop on Embedded Systems*, p. 181--188, 2001.
[119] S. Kumar et al., “A network on chip architecture and design methodology,” in *isvlsi*, 2002, p. 0117.
[120] G. de Micheli and L. Benini, “Networks on Chip: A New Paradigm for Systems on Chip Design,” *Proceedings of the conference on Design, automation and test in Europe*, p. 418–, 2002.
[121] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, “Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives,” *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 28, no. 1, pp. 3-21, 2009.
[122] C. Radu, “Optimized Algorithms for Network-on-Chip Application Mapping,” PhD thesis, “Lucian Blaga” University of Sibiu, Romania, Sibiu, Romania, 2011.
[123] J. Hu and R. Marculescu, “Energy-aware mapping for tile-based NoC architectures under performance constraints,” in *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, Kitakyushu, Japan, 2003, pp. 233-239.
[124] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman & Co. New York, NY, USA, 1979.
[125] J. Hu and R. Marculescu, “Energy- and performance-aware mapping for regular NoC architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, p. 551--562, 2005.
[126] C. Radu and L. Vinţan, “Optimized Simulated Annealing for Network-on-Chip Application Mapping,” in *Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS-18), Bucharest, Romania*, Bucharest, Romania, 2011, vol. 1, pp. 452–459.
[127] “The ns-3 network simulator website.” [Online]. Available: http://www.nsnam.org.
[128] C. Radu and L. Vinţan, “Optimizing Application Mapping Algorithms for NoCs through a Unified Framework,” in *Roedunet International Conference (RoEduNet), 2010 9th*, Sibiu, Romania, 2010, pp. 259 – 264.
[129] 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.

[130] "The Embedded System Synthesis Benchmarks Suite (E3S) website." [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s/.

[131] A.-H. Liu and R. P. Dick, "Automatic run-time extraction of communication graphs from multithreaded applications," in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, Seoul, Korea, 2006, pp. 46-51.

[132] M. M. Kim, J. D. Davis, M. Oskin, and T. Austin, "Polymorphic On-Chip Networks," *SIGARCH Comput. Archit. News*, vol. 36, no. 3, pp. 101-112, 2008.

[133] Jingcao Hu, Umit Y. Ogras, and Radu Marculescu, "System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 12, pp. 2919-2933, 2006.

[134] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "×pipesCompiler: a tool for instantiating application specific networks on chip," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, Paris, France, pp. 884-889.

[135] 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.

[136] S. Uhrig, "The MANy JAva Core processor (MANJAC)," in *HPCS*, 2010, p. 188.

[137] 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.

[138] ***H. Calborean***, "Introduction to the MANJAC system," Computer Science Department, "Lucian Blaga" University of Sibiu, Sibiu, Romania, 4, 2011.