"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

Mulțumiri

În primul rând îi mulțumesc conducătorului meu de doctorat, domnul profesor univ. dr. ing. *Lucian Vinţan*, pentru încrederea acordată şi pentru oportunitatea de a lucra în acest proiect de cercetare. Mulțumiri 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 mulțumesc domnului profesor dr. *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 în mod special colaborarea dintre mine şi membrii acestei echipe, care a continuat şi după întoarcerea mea în țară. Doresc să-i mulțumesc pe această cale lui *Ralf Jahr* împreună cu care am obținut rezultate deosebite şi am ajuns să devenim prieteni.

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

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

Aş dori să îi mulțumesc domnului conferențiar 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 mulțumiri echipei conduse de domnul profesor univ. dr. ing. *Nicolae Ţăpuş* pentru că mi-a permis accesul la sistemul HPC din cadrul Universității Politehnice din Bucureşti. Îi menționez pe această cale pe domnul conferențiar univ. dr. ing. *Emil Slusanschi* şi pe domnul asistent univ. *Alexandru Herişanu* cărora le mulțumesc pentru ajutorul oferit.

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

În cele din urmă vreau să mulțumesc câtorva persoane foarte dragi mie: soției mele *Angela* şi părinților 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 competitivității doctoranzilor într-o Europă unită* cofinanțat din Fondul Social European prin Programul Operațional Sectorial Dezvoltarea Resurselor Umane 2007 - 2013.

Cuprins

Articolele Autorului

Articole publicate

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 în 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 îşi 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 în 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 în 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 în Simulation / Simulation în Education I).

Articole Trimise

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).

Workshop-uri

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

Referate

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 în 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 Introducere

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 țină cont de o mulțime de parametri astfel încât spațiul de proiectare al ansamblului microprocesor-compilator poate ajunge la milioane de miliarde de configurații posibile. Un microprocesor nu poate intra în producție până când nu este evaluat pentru a îndeplini criteriile de performanță şi corectitudine. Evaluarea unei singure configurații poate dura ore sau chiar zile, prin urmare, o evaluare exhaustivă devine imposibilă.

Abordarea actuală este de a utiliza experți umani care selectează configurații, 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 configurații bune devine tot mai grea. Problema este exacerbată de faptul că mai multe obiective trebuie optimizate simultan: performanța, consumul de putere, aria de integrare etc. Descoperirea relațiilor între parametrii unei arhitecturi de calcul şi modul în care influențează ele obiectivele se dovedeşte a fi o sarcină dificilă, aproape imposibil de realizat manual.

O soluție la această problemă este utilizarea de instrumente care efectuează automat o explorare a spațiului de proiectare, folosind diferiți algoritmi euristici de căutare. În viziunea HIPEAC [5], explorarea automată a spațiului configurațiilor (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 în domeniu, aceştia fiind folosiți pentru rezolvarea de probleme NP-hard, dar în ultima perioadă li se acordă un interes tot mai mare, fiind una din puținele soluții viabile pentru DSE.

Scopul acestei teze este de a explora automat spațiul de proiectare a unor arhitecturi de calcul şi de a îmbunătăți algoritmii utilizați prin cunoştințe avansate de domeniu. Pentru aceasta trebuie îndeplinite următoarele **obiective**:

- Analizarea algoritmilor euristici multi-obiectiv folosiți pentru DSE;
- Analizarea comportamentului acestora în probleme reale (în cazul în care există constrângeri între parametri);
- Găsirea unor metrici de evaluare a performanțelor acestor algoritmi;
- Dezvoltarea unei aplicații software robuste şi rapide, care să se poată conecta la orice simulator de arhitecturi de calcul existent;
- Integrarea mai multor algoritmi euristici în această aplicație astfel încât să se poată realiza cu uşurință o comparare a acestora;
- Cercetarea modului în care cunoştințele de domeniu pot fi integrate în aplicația noastră şi cum pot ele influența performanța algoritmilor euristici;
- Testarea şi evaluarea algoritmilor pe diferite tipuri de arhitecturi (single-core, multi-core, System on Chip etc.);
- Compararea algoritmilor şi determinarea impactului integrării cunoştințelor de domeniu asupra rezultatelor obținute.

Capitolul 2 prezintă câțiva algoritmi de căutare bine cunoscuți, frecvent utilizați în probleme de optimizare multi-obiectiv, pe care i-am integrat în aplicația noastră;

Capitolul 3 descrie principalele caracteristici ale uneltei software dezvoltată pentru explorarea automată a spațiului 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 configurațiilor. Aplicația integrează şi o bază de date care îi permite reutilizarea rezultatelor obținute anterior (indivizi deja simulați), conducând deci la o scădere a timpului necesar pentru explorare.

În Capitolul 4 introducem o metodă nouă de accelerare a procesului de explorare şi de creştere a calității rezultatelor. Toți algoritmii de căutare folosiți în această lucrare sunt algoritmi generali şi pot fi folosiți pentru aproape orice problemă de căutare. I-am modificat cu scopul de a utiliza informații 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 1 mai mare de 256KB este „mare", pentru valori sub 64KB este „mic" etc.) şi apoi să introducă reguli uşor de înțeles, cum ar fi: dacă nivelul 1 de cache este mic, atunci nivelul 2 de cache trebuie să fie mare. Informațiile furnizate prin aceste reguli sunt luate în considerare în timpul procesului de explorare, pentru a ghida căutarea. Pe lângă această metodă, mai propunem şi alte tehnici prin care cunoştințele 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 configurații 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 spațiile de proiectare extrem de mari generate de parametrii simulatoarelor. Am realizat o comparație între diferite tipuri de algoritmi de căutare pentru a determina care algoritm oferă rezultatele cele mai bune. Mai prezentăm şi o metodă de a obține 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 ştiințifică a domnului Profesor Lucian Vințan) 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 toți parametrii şi am găsit configurații mult mai bune pentru procesorul (*Alpha*) simulat. Am încercat diferite metode pentru a accelera procesul de DSE. Una dintre ele este introducerea în populația inițială a celor mai bune configurații găsite manual, pornind astfel de la o poziție mai bună în spațiul de căutare. O altă metodă este utilizarea de reguli fuzzy dezvoltate pe baza observațiilor 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 ştiințific Prof. Lucian Vințan), denumit UniMap. Am testat mai mulți algoritmi DSE pentru a afla care obține rezultatele cele mai bune. Am obținut rezultate diferite față de cele din comparațiile 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, definiții, glosar) este strict subordonată scopului şi obiectivelor practice ale acestei teze. Cu alte cuvinte, nu am încercat o prezentare exhaustivă şi nici nu am pretenția unei rigori general valabile.

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

Malcolm Forbes

2 Algoritmi de Căutare

După cum spuneam în capitolul 1, problema DSE-ului este una NP-hard. DSE-ul manual nu este fezabil şi sunt necesare noi metode de căutare. Una dintre ele este să folosim algoritmi care să realizeze această sarcină automat. În acest capitol prezentăm câțiva algoritmi care pot fi utilizați pentru DSE. Ne vom focaliza doar pe cei folosiți de noi în experimentele noastre.

2.1 Optimizare Multi-Obiectiv

În Figura 2.2-1 poate fi văzut frontul Pareto pentru o problemă de minimizare cu două obiective. Punctele reprezintă indivizi în spațiul obiectivelor. Punctele *a* şi *b* sunt denumite puncte nedominate pentru că nu există nici un alt punct care să aibă ambele obiective mai bune decât ele. Cele două puncte nu se domină unul pe altul pentru că punctul *a* este mai bun pe obiectivul 1 dar mai rău pe obiectivul 2 decât punctul *b*. Punctul *c* este dominat de aceste două puncte.

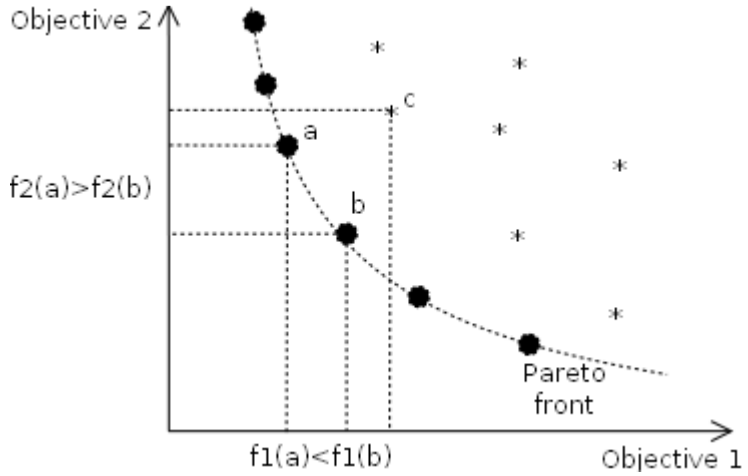

Figura 2.1-1 Pareto front

Problema cu eficienta Pareto este că nu se poate stabili o ordine între indivizii nedominați.

Următoarele definiții sunt utilizate în [5]:

Definiția 1: fiind dați doi vectori $\vec{x}, \vec{y} \in R^n$ spunem că $\vec{x} \le \vec{y}$ dacă $x_i \le y_i$ pentru orice $i = 1,..,n$ şi ca $\vec{x}$ domină $\vec{y}$ (scris ca $\vec{x} \prec \vec{y}$) dacă $\vec{x} \le \vec{y}$ şi $\vec{x} \ne \vec{y}$.

Definiția 2: spunem ca un vector de variabile de decizie $\vec{x} \in X \subset R^n$ este nedominat față de *X*, dacă nu există alt $\vec{x}' \in X$ astfel încât $\vec{f}(\vec{x}') \prec \vec{f}(\vec{x})$, unde $\vec{f}$ este funcția multi-obiectivă.

Definiția 3: un vector de variabile de decizie $\vec{x}^* \in F \subset R^n$ (F este regiunea fezabilă) este Pareto-optimal dacă este nedominat față de *F*.

Definiția 4: Pareto Optimal Set-ul P^* este definit ca:

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

Definiția 5: Pareto Frontul PF^* se defineşte ca:

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

Numim *aproximarea frontului Pareto* (sau *frontul Pareto cunoscut*) soluțiile Pareto optimale în spațiul obiectivelor găsite de un algoritm de DSE. Indivizii, care

fac parte din frontul Pareto aproximativ, alcătuiesc în spaţiul parametrilor *setul Pareto*.

2.1.1 SEMO şi FEMO

În primele experimente am folosit doi algoritmi simpli evolutivi: SEMO şi FEMO [6]. SEMO stochează o arhivă a tuturor indivizilor nedominaţi. Această arhivă reprezintă populaţia curentă. Din această populaţie este ales un părinte. Asupra acestui părinte se aplică mutaţia şi este produs un copil. Noul individ este acceptat în arhivă în cazul în care nu este dominat de către alţi indivizi şi nu există alţi indivizi cu aceleaşi valori pentru obiective. Dacă există în arhivă indivizi dominaţi de noul individ, aceştia sunt eliminaţi. FEMO este o evoluţie a lui SEMO. Diferenţa între ei este că: atunci când se selectează un părinte, algoritmul alege determinist individul cu cel mai mic număr de urmaşi.

Am testat aceşti doi algoritmi pe două probleme de test sintetice LOTZ [6] şi DTLZ1 din familia DTLZ de probleme [7]. Am ajuns la concluzia că ambii algoritmi au reuşit să rezolve problema LOTZ într-o perioadă relativ scurtă de timp (2%, 1% din efortul necesar pentru o evaluare exhaustivă pentru SEMO şi, respectiv, FEMO). Problema DTLZ1 nu a fost soluţionată de nici unul dintre algoritmi (nu a fost găsit nici un punct optim Pareto). Aceste rezultate au fost publicate de noi în [8].

2.1.2 NSGA-II

NSGA-II este un algoritm genetic dezvoltat de Deb et al. [9]. Algoritmul generează o populaţie iniţială pe care o evaluează. Această populaţie iniţială devine populaţia părinte şi pe ea se aplică operatorii de crossover şi mutaţia pentru a obţine populaţia de copii. Copiii sunt evaluaţi. Cele două populaţii sunt unite într-una singură şi sortate în funcţie de relaţia dominantă. Indivizii nedominaţi sunt în continuare sortaţi în funcţie de densitatea zonei în care se află în spaţiul obiectivelor. Cei mai buni dintre ei devin noua populaţie de părinţi şi procesul se repetă.

2.1.3 SPEA2

SPEA2 introdus de Zitzler et al. [10] este alt algoritm genetic. Singurele diferenţe între NSGA-II şi SPEA2 sunt: (a) SPEA2 foloseşte o populaţie externă (arhivă) pentru a păstra indivizii nedominaţi şi (b) atribuie valoarea de fitness într-un mod diferit.

2.1.4 Comparaţie între NSGA-II şi SPEA2

În teză am realizat o comparaţie din punct de vedere teoretic a celor doi algoritmi. Ne-am focalizat în special pe modul de asignare a fitness-ului şi pe rolul arhivei folosite de SPEA2. Am ajuns la concluzia că SPEA2 va tinde să producă mai mulţi indivizi duplicaţi în timpul execuţiei. Acest lucru va duce la o diminuare a numărului unic de indivizi din care algoritmul poate alege părinţi, deci posibil la o convergenţă mai lentă. Ca şi avantaj putem menţiona că dacă indivizii produşi au mai fost evaluaţi o dată în trecut ar putea fi refolosiţi dintr-o bază de date. Aceste supoziţii ne-au fost confirmate şi de rezultatele experimentale.

2.2 Particle Swarm Optimization Multi-Obiectiv

În acest capitol vom prezenta câţiva algoritmi bio-inspiraţi. Aceştia se bazează pe modul în care păsările caută mâncare.

2.2.1 OMOPSO şi SMPSO

OMOPSO [11] este un algoritm multi-obiectiv de optimizare de tip particle swarm optimization (PSO). În aceşti algoritmi populaţia se numeşte swarm (roi). Indivizii sunt numiţi particule. Aceste particule zboară prin spaţiul de căutare urmărind cele mai performante particule la acel moment. Poziţia unei particule este dată de valorile curente ale parametrilor acesteia. Fiecare particulă încearcă să se apropie de particule mai bune. Pentru a obţine acest lucru parametrii săi sunt modificaţi. Schimbarea ia în considerare atât cea mai bună particulă globală (liderul) de la momentul actual, dar şi cea mai bună poziţie pe care a avut-o particula curentă în istoria ei (personal best). După această schimbare particula va avea o nouă poziţie şi trebuie să fie evaluată din nou. După ce toate particulele au fost evaluate, un nou lider este selectat, personal best-ul fiecărei particule este actualizat, iar procesul este repornit. În algoritmii multi-obiectiv pot fi mai mulţi lideri la un anumit moment de timp. Selectarea unui lider este similară cu selectarea părinţilor în NSGA-II.

SMPSO [12] este o dezvoltare a lui OMOPSO. Cea mai importantă diferenţă este că în SMPSO vitezele maxime pe care o particulă le poate atinge sunt limitate.

2.3 Tratarea Constrângerilor

Metoda pe care am folosit-o pentru tratarea constrângerii a fost propusă în [9] pentru a fi folosită împreună cu selecţia binară de timp turneu. Soluţia adoptată este:

- În cazul în care ambii indivizi sunt fezabili, atunci cel cu fitness-ul cel mai bun este selectat;
- În cazul în care un individ este fezabil şi celălalt nu este fezabil, cel fezabil este selectat;
- În cazul în care ambii indivizi sunt nefezabili, este selectat cel care încalcă cele mai puţine constrângeri. Dacă acest lucru nu poate fi determinat iar ambii indivizi încalcă acelaşi număr de constrângeri, individul este ales aleator.

2.4 Metrici de Evaluare a Performantelor Algoritmilor Multi-Obiectiv

2.4.1 Hipervolumul

Această metrică a fost folosită de Zitzler şi Thiele în [13] şi de Coelo în [14]. Fie $X' = (x_1, x_2, ..., x_k) \subseteq X$ un set de vectori de decizie (indivizi). Funcţia $S(X')$ calculează volumul închis între suprafaţa Pareto şi axele de coordonate în spaţiul obiectivelor. Când avem o problemă de minimizare trebuie stabilit un punct în spaţiu (hypervolume reference point – HV),

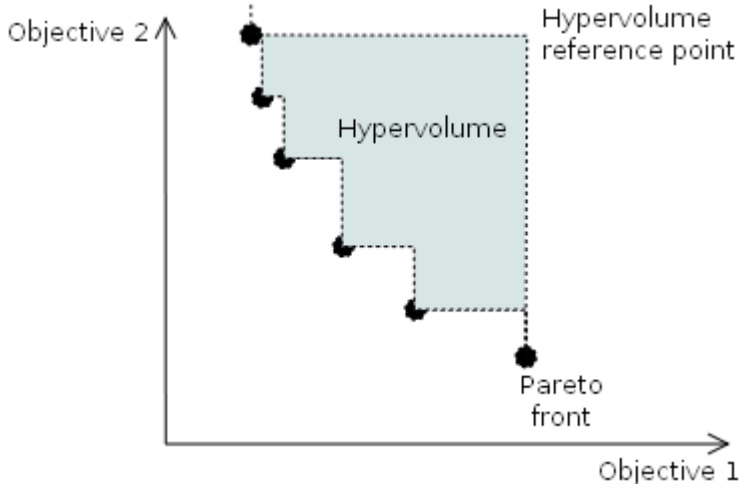

Figura 2.4-1 Calculul hipervolumului pentru o problemă de minimizare

care va înlocui originea axelor în calcularea hipervolumului.

Coordonatele lui HV vor fi date de valorile maxime ale obiectivelor (vezi Figura 2.4-1). Valoarea hipervolumului reprezintă procentul acoperit de volumul cuprins între punctul de referința HV şi suprafața Pareto, din volumul total cuprins între punctul de referință şi axe (valorile sunt normalizate).

2.4.2 Two Set Difference Hypervolume

Această metrică a fost propusă de E. Zitzler în teza sa de doctorat [15]. Two Set Difference Hypervolume (TSDH) se defineşte ca:

$$TSDH(X', X'') = H(X'+X'') - H(X'')$$

unde $X', X'' \subseteq X$ sunt două seturi de vectori de decizie, $H(X)$ este valoarea hipervolumului acoperit de X, şi $X'+X''$ este setul de vectori de decizie nedominați obținut după uniunea dintre X' şi X''.

$TSDH(X', X'')$ calculează hipervolumul spațiului acoperit de X' dar nu şi de X''.

2.4.3 Coverage of Two Sets

Această metrică a fost folosită de Zitzler şi Thiele [13] şi Palermo [16]. Ea calculează procentajul de indivizi dintr-o populație dominați de indivizi din altă populație.

Fie $X', X'' \subseteq X$ două seturi de vectori de decizie. Coverage-ul se calculează astfel:

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

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

Albert Einstein

3 FADSE: Un Framework pentru Automatic Design Space Exploration

Framework for Automatic Design Space Exploration (FADSE) este un utilitar, dezvoltat de noi, care este capabil să efectueze automat DSE folosind o mare varietate de algoritmi. FADSE este capabil să ruleze evaluările într-o manieră distribuită pe rețele locale (LAN) sau sisteme HPC (High Performance Computing). Este un framework robust. Își poate reveni din situații ca: probleme cu rețeaua de interconectare, clienți care nu mai răspund, până la pierderea totală a curentului electric.

FADSE permite utilizatorului să își exprime cunoştințele legate de problema de rezolvat intr-un limbaj cât mai natural. Frameworkul nostru va prelua aceste informații, le va interpreta şi le va transmite algoritmilor de DSE, care vor ține cont de ele.

Pentru a avea acces la cât mai mulți algoritmi performanți de DSE am integrat în FADSE librăria jMetal [17]. Am extins această librărie şi acum putem face DSE distribuit folosind diferite simulatoare: Multi2Sim [18], GAP [19], M5 (http://www.m5sim.org) şi M-SIM (http://www.cs.binghamton.edu/~msim/). Am inclus funcții de test sintetice (LOTZ) şi metrici ca: error ratio, coverage of two sets, hypervolume, hypervolume two set difference.

FADSE este dezvoltat în JAVA ceea ce ne permite să îl rulăm pe o mulțime de sisteme.

Ultima versiune a lui FADSE poate fi obținută de la adresa: http://code.google.com/p/fadse/.

3.1 Accelerarea Procesului de DSE prin Evaluare Distribuită

FADSE a fost conceput ca o aplicație client-server. Serverul rulează algoritmul de DSE şi clienții efectuează simulări.

Am modificat o parte din algoritmii disponibili în librăria jMetal pentru a permite o evaluare distribuită a configurațiilor. Se ştie că majoritatea algoritmilor evolutivi/bio-inspirați pot fi paralelizați uşor. Ei evaluează o populație de mai mulți indivizi şi doar după ce toate evaluările sunt terminate folosesc rezultatele obținute. Profitând de acest comportament al procesului de evaluare algoritmii pot fi distribuiți cu uşurință. FADSE evaluează în paralel copiii obținuți în urma proceselor de mutație si/sau crossover (vezi Figura 3.1 1). Dacă un individ trebuie evaluat pe mai multe benchmark-uri atunci şi acest proces este distribuit. Ca un exemplu: un algoritm cu 100 de indivizi în populație, unde fiecare individ trebuie evaluat pe 10 benchmark-uri duce la 1000 de simulări care se pot face în paralel. Din toate utilitarele de DSE pe care le-am analizat nici unul nu oferă posibilitatea de a evalua configurațiile în paralel.

FADSE este proiectat pentru a face față problemelor apărute la clienți sau în rețea. Dacă un client nu răspunde, evaluarea, care a fost programată pe el, este trimisă la un alt client. După o serie de încercări, individul este marcat ca infezabil în cazul în care nici unul dintre clienți nu reuşeşte să-l evalueze. Acest lucru îi permite explorării să continue, chiar dacă unele configurații s-au dovedit imposibil de simulat. Chiar dacă întregul sistem trebuie să fie repornit (serverul are probleme, s-a întrerupt alimentarea de la rețeaua de curent), FADSE poate reporni de la un punct recent.

Pentru aceasta am implementat un mecanism de checkpointing. În plus clienții au mecanism de recuperare implementat. Când un client este pornit, în paralel cu el este pornit şi un watchdog timer. Acesta monitorizează clientul şi dacă acesta nu primeşte nici un mesaj pentru o perioadă lungă de timp, restartează clientul (inclusiv maşina virtuală JAVA). Acest comportament ne permite să evităm unele probleme care pot apărea pe clienți.

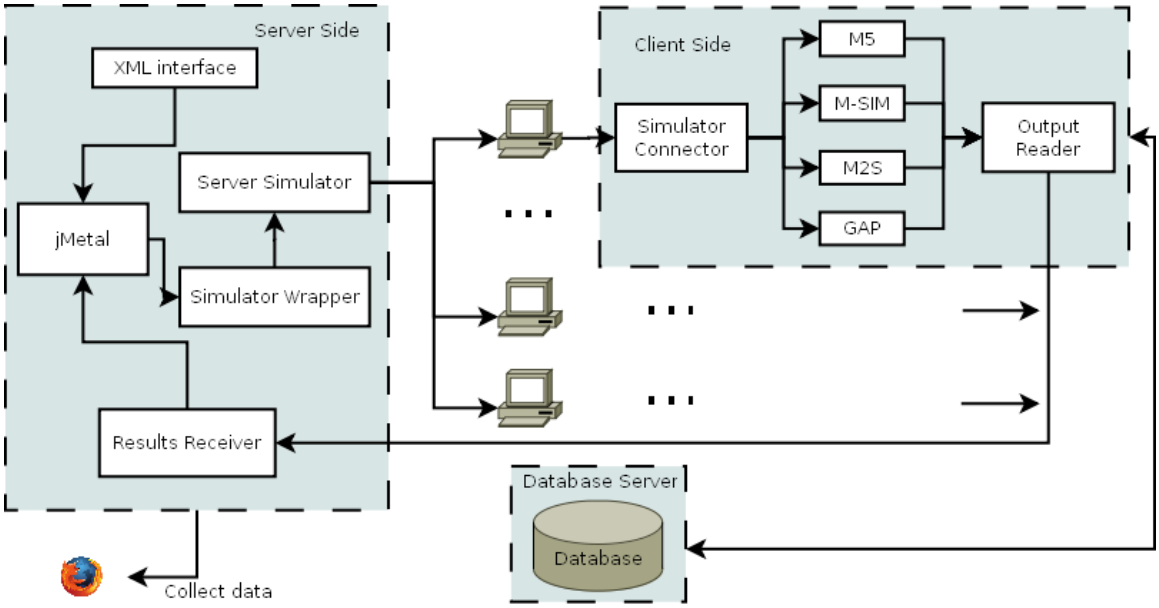

Figura 3.1-1 Structura lui FADSE (versiunea distribuita)

Cu această configurație client-server am fost capabili să rulăm FADSE pe rețele locale, sisteme HPC şi maşini virtuale multi-procesor. O parte dintre sistemele pe care am testat FADSE sunt prezentate mai jos.

Am testat FADSE pe o rețea locală (LAN) cu 9 maşini Intel dual-core. Ele au fost configurate într-un VPN (Virtual Private Network) pentru a ne asigura ca IP-urile lor nu se schimbă în timpul rulării (IP-urile se alocau cu ajutorul unui server DHCP şi căderile de curent erau frecvente). Un computer a fost ales ca server. Pe acesta rula algoritmul de DSE şi serverul de MySQL (vezi Capitolul 3.2 cu privire la sistemul de reutilizare a rezultatelor implementat în FADSE). Pe toate celelalte maşini s-au pornit câte doi clienți FADSE. Acest sistem a fost folosit pentru a obține unele dintre rezultatele prezentate în Capitolul 5. Pe maşinile din sistem rula atât Windows XP cât şi Windows 7.

FADSE a fost rulat pe două sisteme HPC: unul situat la Universitatea "Lucian Blaga" din Sibiu şi unul în Bucureşti, la Universitatea Politehnică. Cel de la Sibiu a fost utilizat pe scară largă. Toate rezultatele prezentate în capitolele 6 şi 7 au fost obținute cu acest sistem. Ambele clustere folosesc Red Hat Linux ca sistem de operare.

Sistemul HPC de la Sibiu (http://zamolxe.hpc.ulbsibiu.ro/ganglia/) conține două clustere: unul bazat pe procesoare Intel Xeon quad core (CSAC cluster) şi unul bazat pe procesoare IBM Cell (ACAPS cluster).

Cluster-ul CSAC conține 15 blade-uri, fiecare cu câte două procesoare şi 4 GB de DRAM. Aceasta înseamnă că sunt 120 de nuclee în întregul sistem.

Cluster-ul ACAPS conține două blade-uri, fiecare blade cu câte două procesoare Cell. Fiecare procesor Cell conține un nucleu PowerPC multi-thread şi 9

unități specializate de procesare. Am testat cu succes FADSE şi pe acest sistem. Am pornit un server de pe un nod din cluster-ul CSAC şi clienții pe cluster-ul ACAPS.

Testele au fost efectuate cu succes pe sistemul HPC de la Universitatea Politehnică din Bucureşti. Acest sistem permite pornirea proceselor printr-un sistem de job-uri. Au fost create script-uri speciale pentru a rula FADSE pe acest sistem. În viitor ne propunem să integrăm în FADSE posibilitatea de a utiliza comenzi de tip batch direct din conectori. Testele au fost efectuate folosind în jur de 100 de clienți, care rulau atât pe procesoare Nehalem cât şi pe procesoare Opteron.

Prezentăm un ultim caz: am rulat FADSE pe o maşină virtuală Windows 7, cu 32 de procesoare de la Universitatea din Augsburg. Am folosit această maşină pentru a obține cea mai mare parte a rezultatelor din Capitolul 5. Am rulat fie un singur server cu 32 de clienți (serverul nu consumă mult timp la procesor, iar acesta este inactiv de cele mai multe ori în timpul de simulare), sau două servere, în paralel, cu câte 16 clienți fiecare.

Mai multe servere pot fi pornite pe aceeaşi maşină şi pot rula, de asemenea, alături de clienți. Acest lucru înseamnă că mai multe procese de DSE pot fi începute în paralel pe acelaşi sistem şi că toate resursele disponibile pot fi folosite.

3.2 Accelerarea Procesului de DSE prin Reutilizarea Rezultatelor

Algoritmi de DSE tind să producă aceiaşi indivizi după câteva generații. În loc să îi mai simulăm o dată am putea refolosi rezultatele obținute anterior dintr-o bază de date. Pentru aceasta am conectat FADSE cu un sistem de gestionare a bazelor de date (DBMS).

Integrarea cu baza de date s-a dovedit a fi de succes. Am ajuns la o reutilizare de aproximativ 67% în timpul unei rulări cu 100 de generații cu o populație de 100 de indivizi. Ceea ce înseamnă o reducere importantă a timpului necesar explorării.

3.3 Interfața Universală – Conectori

FADSE este conceput în aşa fel încât să poată fi conectat la aproape orice simulator existent, cu un efort minim şi, în majoritatea situațiilor, cu nici o modificare în cadrul simulatorului (codul sursă nu este necesar).

Pentru aceasta utilizatorul trebuie să implementeze în cadrul FADSE un conector. Acesta va prelua parametrii dați de FADSE şi va porni cu ei simulatorul.

3.4 Interfața Extensibilă de Configurare

O interfață XML este utilizată pentru a configura FADSE. Aici se specifică spațiul de proiectare şi constrângeri specifice fiecărei probleme. În primul rând, utilizatorul trebuie să specifice conectorul simulatorului şi setul de parametri ceruți de conector, după care se specifică o listă de benchmark-uri, conexiunea la baza de date, lista de parametri care trebuie variați (aceşti parametrii pot fi de tip întreg, progresii aritmetice/geometrice, liste de string-uri, etc.), obiectivele care trebuie optimizate.

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

Albert Einstein

4 Îmbunătățirea lui FADSE cu Cunoştințe de Domeniu

Toți algoritmii incluşi în FADSE sunt generali. Ei au fost concepuți pentru a rezolva clase întregi de probleme. Algoritmii specializați, care includ cunoştințe despre problema care trebuie să fie rezolvată, ar putea oferi rezultate mai bune, dar aceştia pot fi aplicați la un număr limitat de probleme. Ne dorim ca FADSE să fie un cadru general, un instrument care poate fi folosit cu toate simulatoarele existente. Scopul acestui capitol este de a identifica metode prin care un utilizator îşi poate exprima cunoştințele într-un mod simplu şi cum le putem include apoi în FADSE, fără a pierde din generalitate.

4.1 Constrângerea Spațiului de Proiectare

Constrângerile sunt necesare atunci când optimizăm arhitecturi de procesoare pentru a evita configurații imposibile sau configurații care proiectantul ştie că nu vor duce la rezultate foarte bune. Unul dintre cele mai bune exemple este că dimensiunea unui cache de nivel 2 trebuie să fie mai mare decât dimensiunea nivelului 1 de cache. Pentru un algoritm DSE, parametrul „dimensiune cache" nu are nici un sens şi astfel s-ar putea genera configurații în care regula de mai sus nu este respectată. Aceste situații pot fi evitate prin utilizarea de constrângeri.

În proiectarea interfeței XML pentru FADSE, prin care utilizatorul poate specifica constrângerile, am folosit ca model instrumentul M3Explorer [20], dar implementarea este originală. Migrarea de la M3Explorer la FADSE nu ar trebui să fie dificilă, deoarece interfața FADSE este în cea mai mare parte un superset al interfeței M3Explorer.

Constrângerile implementate în FADSE sunt unele dintre cele mai puternice mecanisme prin care utilizatorul poate influența procesul de explorare. Prin ele se poate controla dimensiunea spațiului şi granițele lui. Constrângerile ajută algoritmul să evite explorarea zonelor neinteresante, ducând la un proces de explorare mult mai rapid. Ele au fost folosite intensiv în explorările noastre (in special în cele din Capitolul 6).

4.2 Parametri Ierarhici

4.2.1 Motivație

În multe situații întâlnite în optimizarea arhitecturilor de calculatoare există parametri care au sens doar în legătură cu alți parametri. Un exemplu foarte simplu este când în arhitectură putem alege între mai multe tipuri de predictoare de branch-uri: predictoare neurale, predictoare two-level, etc. Este evident că, dacă la un moment dat, arhitectura noastră conține un predictor de tip neural, parametrii predictorului two level nu au sens, ei nu trebuie variați ci trebuie invalidați.

Pentru a rezolva această problemă am propus şi am implementat o interfață XML care îi permite utilizatorului să descrie astfel de relații (ierarhii) între parametrii. Informația dată în XML trebuie injectată în algoritmul de căutare. Pentru aceasta am propus operatori noi de crossover şi mutație. Aceşti noi operatori sunt în stare să identifice punctele valide de mutație/crossover şi să evite producerea de indivizi cu genotip diferit, dar cu acelaşi fenotip.

4.2.2 Adaptarea Operatorilor Genetici

Operatorii folosesc arbori care conțin informația de (in)validitate a parametrilor.

4.2.2.1 Crossover

Operatorul de crossover primeşte doi arbori şi doi indivizi. Foloseşte aceşti arbori pentru a trece la o reprezentare arborescentă a indivizilor. Din această reprezentare extrage câte o mască pentru fiecare individ, care marchează parametrii valizi. Folosind principiul super poziției din cele două măşti se obțin punctele valide unde se poate aplica crossoverul. Crossoverul se poate face pe muchii ale arborelui care nu pointează către noduri invalide.

4.2.2.2 Mutație

Operatorul de mutație este mai simplu, foloseşte un singur arbore. La fel ca la crossover se extrage o mască din arbore care precizează parametrii valizi pentru mutație. Unul dintre aceşti parametri este ales şi operatorul de mutație este aplicat (bineînțeles se ține cont de probabilitatea de mutație).

4.3 Introducerea de Cunoştințe de Domeniu prin Reguli Fuzzy

În acest paragraf propunem utilizarea regulilor fuzzy pentru reprezentarea într-un limbaj natural a cunoştințelor de domeniu deținute de arhitectul sistemului. Din ceea ce ştim suntem primii care propun utilizarea regulilor fuzzy ca metodă apriori de descriere a cunoştințelor în problema DSE-ului pentru arhitecturi de calculatoare.

Ele ne permit să scriem reguli de genul:

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

4.3.1 Sistemul de Inferență Mamdani

Dintre sistemele de inferență existente în logica fuzzy ne-am focalizat pe cel propus de Mamdani [21]. Acest lucru înseamnă că am folosit funcția MIN, MAX pentru operatorii AND şi respectiv OR din regulile fuzzy. Pentru implicație am folosit tot formula propusă de Mamdani şi anume operatorul MIN. Pentru agregarea funcțiilor de membership obținute din fiecare regulă, am folosit funcția MAX [22][23]. Pentru defuzificare am folosit metoda centrului de greutate (COG).

4.3.2 Integrarea Regulilor Fuzzy în FADSE

4.3.2.1 Limbajul FCL

Pentru a reprezenta funcțiile de membership şi regulile fuzzy am folosit limbajul FCL (Fuzzy Control Language) [24]. FCL este un limbaj standard publicat de International Electrotechnical Commission (IEC) [25] [24]. Pentru a putea folosi acest limbaj am integrat în FADSE librăria jFuzzyLogic [26]. În plus librăria jFuzzyLogic include şi sistemele de inferența folosite în experimentele noastre. FADSE acceptă ca input fişiere scrise în limbajul FCL.

4.3.2.2 Operatori de Mutație

După ce utilizatorul a specificat regulile, ele sunt aplicate în timpul procesului de DSE. Informația pe care ele o dau trebuie să influențeze algoritmii de căutare. Pentru aceasta am implementat noi operatori genetici (mutație).

4.3.2.2.1 Modificarea Operatorului de Mutație Bit-flip

În această lucrare am extins operatorul de mutație pentru a lua în considerare informația obținută în urma defuzificării rezultatului obținut după aplicarea regulilor.

Operatorul clasic de mutație bit-flip pentru întregi este modificat astfel:

1. Pentru toți parametrii (gene) din individ (cromozom);
   1.1. Dacă există o regulă fuzzy pentru acest parametru;
      1.1.1. Aplică mutația (cu o probabilitate) luând în considerare valoarea obținută din regulile fuzzy;
   1.2. Altfel (aplică mutația bit-flip);
      1.2.1. Generează un număr pseudo-aleator între 0 şi 1;
      1.2.2. Dacă numărul pseudo-aleatoriu este mai mic decât probabilitatea de mutație;
         1.2.2.1. Schimbă valoarea parametrului curent la o valoare pseudo-aleatoare din intervalul lui de definiție;
2. STOP.

Dacă variabila curentă apare ca output la una din regulile definite în fişierul FCL atunci există şansa ca acest parametru să primească valoarea obținută în urma defuzificării.

Vom prezenta două implementări ale mutației. Ele diferă prin modul în care se calculează probabilitatea cu care se aplică informația dată de regulile fuzzy: probabilitate constantă sau distribuție Gaussiană de probabilitate.

4.3.2.2.2 Probabilitate Constantă

Pentru a păstra diversitatea, informația obținută în urma aplicării regulilor fuzzy nu este luată tot timpul în considerare. Pentru a obține acest efect folosim o probabilitate de aplicare a informației, pe care o denumim *fuzzy probability*.

În implementarea simplă această probabilitate este constantă pe timpul rulării algoritmului de DSE şi este egală cu probabilitatea de mutație.

4.3.2.2.3 Distribuție Gaussiană a Probabilității

A doua propunere pentru mutație foloseşte o distribuție Gaussiană a probabilității de aplicare a informației obținută din regulile fuzzy. Scopul a fost de a da o prioritate mai mare la începutul rulării regulilor fuzzy, iar după câteva generații algoritmul să fie lăsat liber să caute. Am ales pentru funcția Gaussian astfel parametrii încât după 500 de indivizi evaluați să tindă spre valoarea probabilității de mutație (din mutația bit-flip). Funcția obținuta este prezentata mai jos

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

unde *x* reprezintă al câtelea individ este trimis operatorului de mutație. Acest x creşte cu o unitate la fiecare evaluare.

Domeniul valorilor obținut în urma calculării funcției va fi în (*mutation_probability*, 1]. Am descoperit în experimentele noastre că nu este bine să forțezi aplicarea acestor reguli cu o probabilitate de 1, altfel există şansa să se piardă din diversitatea populației şi toți indivizii vor reflecta preferințele arhitectului, care pot să nu fie cele mai bune. Pentru a evita astfel de situații am înmulțit valoarea

obținută din funcția de mai sus cu 0.8 pentru a evita valoarea de 1. Astfel probabilitatea maximă va fi de 80%. În plus mai calculăm şi membership-ul μ valorii obținute în urma defuzificării pe funcția de output obținută după agregare. Acest membership îl folosim ca pe o confidentă în valoarea obținută în urma defuzificării.

Funcția finală folosită este:

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

4.3.2.2.4 Defuzificator Aleator

Există situații speciale, când metoda de defuzificare utilizată de noi (centrul de greutate - COG) nu oferă rezultate bune. O astfel de situație poate apărea în cazul în care funcțiile de intrare fuzzy au o formă aproape dreptunghiulară. În această situație, orice valoare de intrare va avea o valoare de membership egală (sau aproape egală) cu 1. Dacă valoarea este tot timpul 1 atunci funcțiile de membership de ieşire vor fi identice pentru toate intrările. Acest lucru înseamnă că COG va avea aceeaşi valoare de fiecare dată. Pentru a evita astfel de situații, am implementat o nouă metodă de defuzificare. Acest nou defuzificator alege aleator o valoare din funcția de membership de ieşire. Am folosit acest defuzificator atunci când regulile utilizate au fost obținute automat.

4.3.2.3 Parametri Virtuali

Arhitecții preferă să-si exprime de multe ori regulile la nivel de module: dimensiunea cache-ului de nivel 1, etc. Dar aceste module sunt definite de mai mulți parametri: număr de linii din cache, dimensiunea blocului, asociativitatea. Pentru ai permite designer-ului să-şi exprime cunoştințele la nivelul de abstractizare dorit, am implementat parametrii virtuali. Aceştia îi permit să definească concepte formate din mai mulți parametri pe care le poate folosi apoi în interiorul regulilor fuzzy.

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

Jan L.A. van de Snepscheu

5 Optimizare Hardware-Software Multi-Obiectivă a lui Grid ALU Processor

În acest capitol am realizat un DSE pe un procesor superscalar dezvoltat la Universitatea din Augsburg denumit GAP (Grid ALU Processor) [27] [28]. În plus am folosit FADSE pentru optimizarea alături de GAP şi a unui optimizator de cod dezvoltat special pentru acest procesor denumit GAPtimize [29] [30] [31].

Parametrii variați în experimente generează un spațiu de căutare de peste un milion de configurații posibile când optimizăm doar GAP-ul. Când am optimizat atât parametrii lui GAP cât şi cei ai lui GAPtimize, în acelaşi timp, spațiul de căutare este de peste 10^{16} configurații posibile.

Am optimizat GAP şi GAPtimize urmărind două obiective: viteza de procesare (CPI) şi complexitatea hardware-ului. Ultimul obiectiv a fost introdus de noi în [29].

Această cercetare a fost rezultatul unei colaborări între noi şi Universitatea din Augsburg prin intermediul domnului Profesor Theo Ungerer şi a studentului doctorand Ralf Jahr. Rezultatele obținute au fost publicate în conferințe [32] [33] [34] (in review) sau prezentate la workshop-uri [35].

5.1 DSE automat pe Parametrii Hardware

În acest paragraf continuăm munca făcută da Basher Shehan în teza lui de doctorat [36]. El a realizat un DSE manual pe configurațiile hardware. Noi încercăm sa vedem dacă putem sa obținem rezultate mai bune decât cele obținute manual printr-un DSE automat folosind FADSE.

Am folosit NSGA-II ca algoritm de explorare [9] şi am optimizat procesorul GAP pe 10 benchmark-uri din suita MiBench [37].

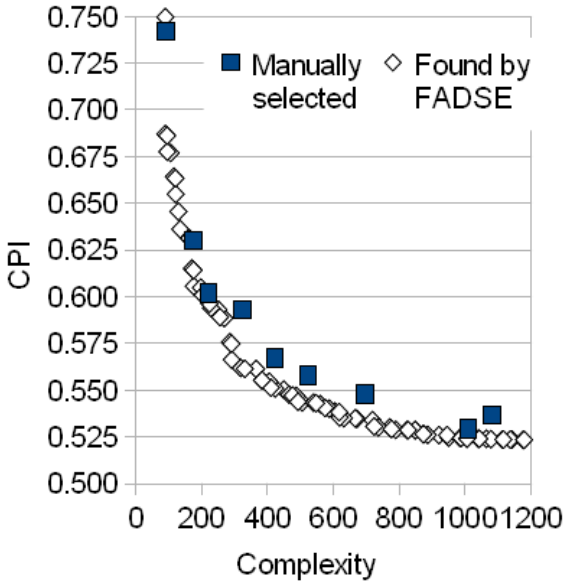

Figura 5.1-1 Comparație între rezultatele obținute manual şi cele obținute automat (doar o porțiune a frontului Pareto este prezentată)

În primele experimente am comparat rulări cu diferite dimensiuni ale populației. Rezultatele au arătat că o populație de dimensiunea 100 obține cele mai bune rezultate.

Următorul pas a fost să realizăm un DSE pe GAP. Rezultatele au fost foarte bune (vezi Figura 5.1-1). Am reuşit să obținem configurații mai bune decât cele găsite manual. Am găsit configurații cu aceeaşi performanță (CPI) dar cu o complexitate hardware de două ori mai mică decât cea a configurațiilor obținute manual.

Am observat că în alegerea manuală a parametrilor designer-ul a avut o preconcepție greşită: a setat numărul de linii şi coloane din matricea de ALU-uri ca egale. Această preconcepție greşită a dus la o creştere nejustificată a complexității hardware fără câştiguri de performanță. Explorarea automată a depăşit aceste probleme şi a obținut rezultate mai bune.

Din punct de vedere al îmbunătățirilor introduse în FADSE am observat o reutilizare de 67% în procesul de explorare în acest experiment. Acest lucru înseamnă că 67% din simulări au fost reutilizate din baza de date ceea ce duce la o reducere masivă a timpului de explorare.

Tot în acest experiment am demonstrat că FADSE găseşte rezultate bune în spații mari de căutare, este robust (poate rula pe perioade îndelungate de timp) şi evaluarea distribuită duce la o reducere importantă a timpului de explorare. Am mai demonstrat că GAP este scalabil şi că un cache mai mare nu anulează efectele matricei de ALU-uri prezente în arhitectură.

5.2 DSE Automat pe Parametrii Hardware şi pe cei ai Compilatorului

În acest paragraf am realizat un DSE al parametrilor hardware în acelaşi timp cu parametrii ai optimizatorului de cod GAPtimize.

Am rulat inițial cu un singur benchmark folosind *function inlining* ca optimizare de cod. Creşterea de performanță obținută față de rulările fără optimizare de cod a fost de 9,1%. După acest pas am introdus toate cele 10 benchmark-uri în optimizare (vezi Figura 5.2-1). FADSE găseşte şi în acest caz rezultate foarte bune.

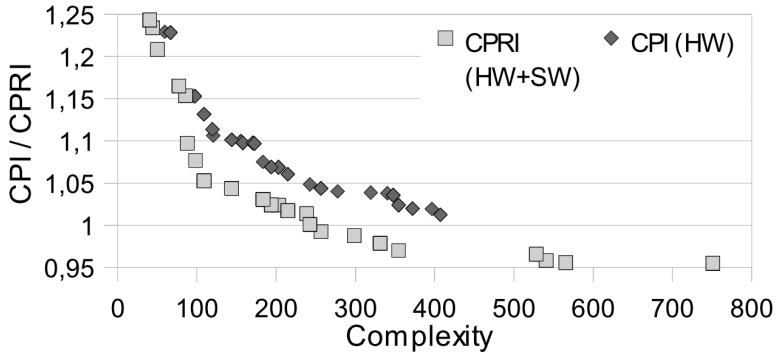

Figura 5.2-1 DSE pe GAP şi GAPtimize

5.3 Comparație între Algoritmii de DSE

Rezultatele prezentate în acest paragraf au fost publicate în articolul intitulat "Optimizing a Superscalar System using Multi-objective Design Space Exploration" [38].

Am realizat o comparație între câțiva dintre algoritmii implementați în FADSE. Am ales NSGA-II, SPEA2 şi SMPSO. I-am comparat atât intr-o explorare pe GAP cât şi intr-un DSE cu GAP şi GAPtimize în acelaşi timp.

Când am rulat FADSE doar pe GAP, SMPSO a obținut cele mai bune rezultate urmat de NSGA-II şi SPEA2 (vezi Figura 5.3-1).

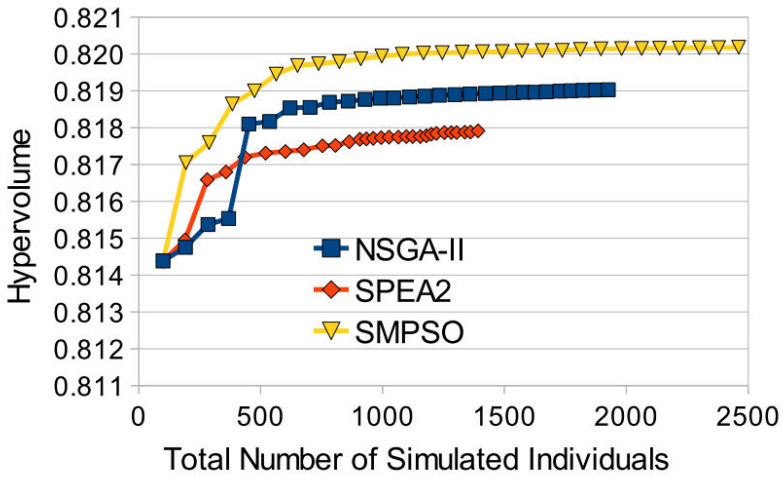

Figura 5.3-1 Valoarea Hipervolumului obținut de cei trei algoritmi în funcție de câți indivizi au fost evaluați

SMPSO trimite cei mai mulți indivizi unici spre evaluare, dar în acelaşi timp obține cele mai bune rezultate. SPEA2 simulează cei mai putini indivizi, dar nu reuşeşte să atingă rezultate atât de bune ca ceilalți doi algoritmi.

Din punct de vedere al convergenței SMPSO este cel mai rapid.

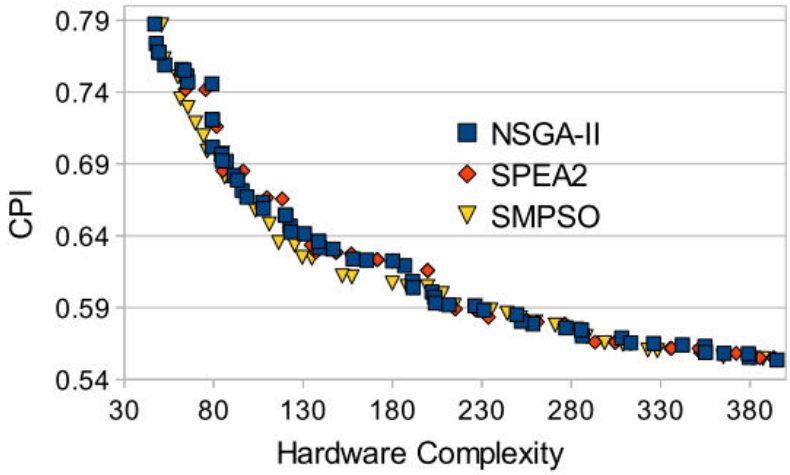

Figura 5.3-2 Comparație între fronturile Pareto obținute de cei trei algoritmi pe GAP

Următorul pas a fost să comparăm fronturile Pareto aproximative obținute de cei trei algoritmi (vezi Figura 5.3-2). Am observat că nu există o mare diferență între rezultatele obținute de algoritmi. Practic soluțiile găsite se suprapun şi diferențele între configurații sunt foarte mici.

În următorul experiment am introdus şi GAPtimize alături de GAP şi am comparat cei trei algoritmi. SMPSO rămâne cel mai performant dintre ei (vezi Figura 5.3-3), dar NSGA-II are rezultate mai slabe decât cele ale lui SPEA2.

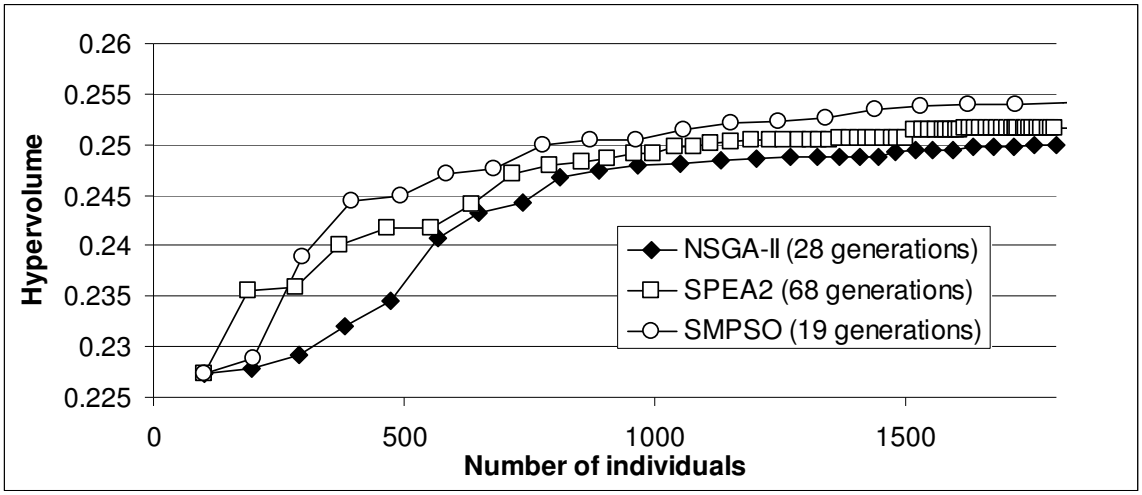

Figura 5.3-3 Valoarea Hipervolumului obținut de cei trei algoritmi pe GAP şi GAPtimize în funcție de câți indivizi au fost evaluați

Am comparat şi fronturile Pareto obținute de cei trei algoritmi pe problema GAP şi GAPtimize.

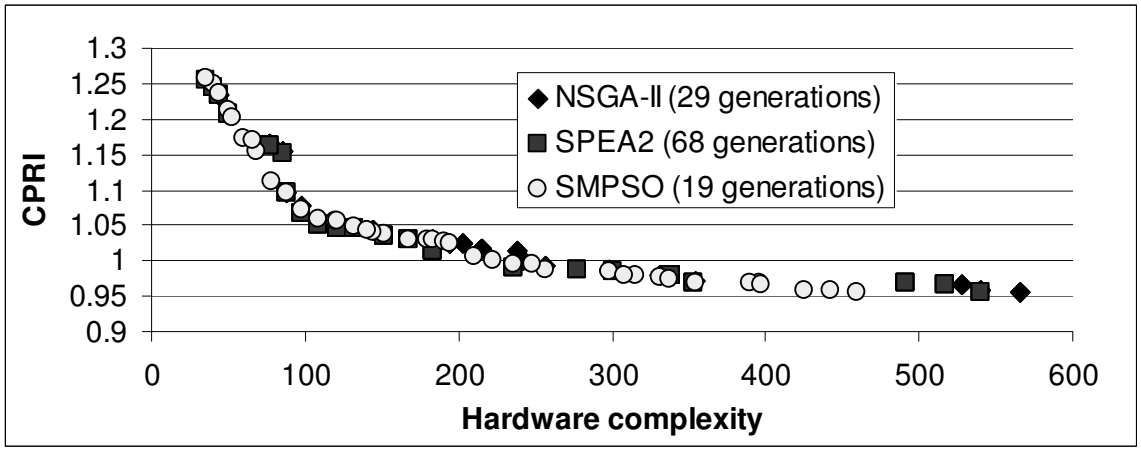

Figura 5.3-4 Comparare între fronturile Pareto aproximative obținute de cei trei algoritmi când am optimizat atât GAP cât şi GAPtimize

Concluzia rămâne aceeaşi diferență între rezultate este infimă.

În experimentele noastre am folosit mai multe metrici: hipervolum, coverage, two set difference hypervolume. Am ajuns la concluzia că coverage-ul poate să fie înşelător în unele cazuri arătând o diferență mare între algoritmi când în realitate rezultatele obținute sunt foarte apropiate. Problema metricii este că nu are nici un prag (treshold) la calculul relației de dominanță. Chiar dacă diferența pe un singur obiectiv este foarte mică, individul este considerat în continuare dominat. Metrici de acest fel au fost propuse în [14]. Hipervolumul ajută utilizatorul să observe viteza de convergență a algoritmilor. Acesta oferă, de asemenea, unele informații cu privire la calitatea soluțiilor atunci când mai mulți algoritmi sunt analizați în aceeaşi figura, toți raportându-se la acelaşi punct de referință (hypervolume reference point). Cu toate acestea, diferența dintre valorile hipervolumului nu ar trebui să fie luate în considerare, deoarece depind puternic de poziția punctului de referință.

Recomandăm ca designer-ul să aleagă cel mai bun algoritm după analiza suprafețelor Pareto obținute.

Analizând fronturile Pareto aproximative obținute de cei trei algoritmi, putem concluziona că diferențele sunt mici între rezultate. Cele mai mari diferențe nu depăşesc 1% în termeni de viteză (CPI), şi de obicei se află între 0,1% şi 0,01%, la aceeaşi complexitate. Diferența dintre algoritmi apare în viteză de convergență, unde algoritmul de particle swarm optimization (SMPSO) obține cele mai bune rezultate. Între algoritmii genetici nu putem spune care este cel mai bun: NSGA-II a avut rezultate mai bune în primul experiment dar mai slabe în al doilea decât SPEA2.

5.4 Reguli Generate Automat din Explorări Anterioare

În acest paragraf propunem o îmbunătățire a lui FADSE care se bazează pe regulile fuzzy prezentate în capitolul 4. Scopul acestei îmbunătățiri este de a obține în mod automat reguli din rezultatele simulărilor anterioare. Ideea este că un utilizator poate rula un proces rapid de DSE pe un singur benchmark şi din rezultatele obținute să extragă cunoştințe pe care să le folosească apoi intr-un DSE de lungă durată, pe mai multe benchmark-uri. O altă situație poate fi atunci când o companie construieşte un procesor şi efectuează un DSE. Un client ar putea veni şi să dorească să optimizeze arhitectura pentru o anumită sarcină, sau producătorul doreşte să adauge un nou modul în design-ul existent. Compania ar putea extrage cunoştințe din explorarea anterioară şi să ofere clientului aceste informații astfel încât el să poată accelera DSE-ul lui.

Rezultatele din acest paragraf au fost trimise spre publicare cu articolul intitulat "Boosting Design Space Explorations with Existing or Automatically Learned Knowledge" [34].

Ideea de bază este utilizarea de tehnici de învățare automată care vor produce arbori de decizie. Datele folosite au fost preluate din explorări anterioare. Din aceşti arbori am extras automat reguli care au fost folosite apoi în FADSE prin mecanismul de reguli fuzzy.

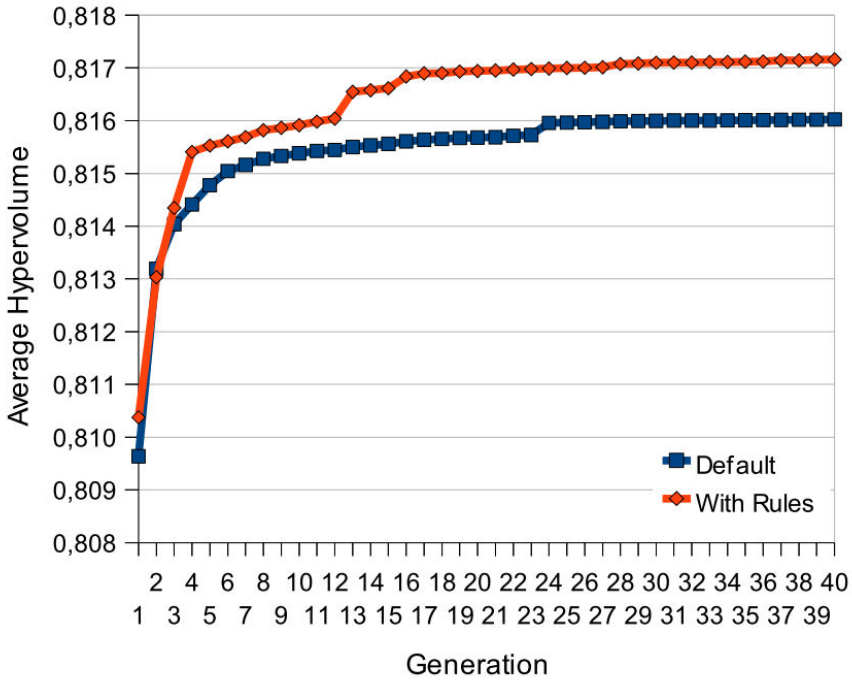

Figura 5.4-1 Comparație între hipervolumurile obținute atât cu cât şi fără reguli obținute automat

Rezultatele obținute au fost foarte bune. În Figura 5.4-1 se vede o astfel de rulare. Regulile au fost generate din datele obținute în urma rulării unui experiment scurt folosind un singur benchmark – stringsearch (unul din cele mai scurte din suita MiBench). Regulile obținute sunt apoi introduse în DSE-ul făcut pe toate benchmark-urile. Rezultatul este: o convergență mai rapidă a rulării cu reguli şi în acelaşi timp rezultate mai bune decât în rularea fără reguli. Hipervolumul obținut cu reguli nu este atins de rularea fără reguli în cele 40 de generații cât am rulat experimentul.

Experimente cu o probabilitate constantă de a aplica regulile au fost efectuate, dar rezultatele nu au fost foarte bune. Se pare că numărul mare de reguli obținute automat, fiecare cu un număr mare de intervale de membership asociat fiecărui termen lingvistic conduce la rezultate foarte bune şi ajută la păstrarea diversității când se foloseşte o distribuție Gaussian de probabilitate pentru aplicarea regulilor. În capitolul 6 am observat că, dacă regulile sunt puține, probabilitatea constantă duce la rezultate mai bune.

5.5 Rularea cu Parametrii Ierarhici

În acest paragraf am testat parametrii ierarhici. Am folosit mai multe optimizări de cod din GAPtimize care aveau fiecare parametrii asociați. Aceste optimizări puteau fi activate sau dezactivate. Până în acest moment nu am rulat pe mai multe benchmark-uri pentru a prezenta rezultate concludente, doar pe un singur benchmark.

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

Alan Saporta

6 Optimizări Multi-Obiectiv ale Parametrilor Arhitecturilor Single-Core şi Multi-Core

În acest capitol continuăm munca făcută de domnul Dr. Árpád Gellért în teza sa de doctorat [39] şi în câteva articole subsecvente [40][41]. Am preluat DSE-ul manual făcut de dânsul pe un procesor Alpha AXP 21264 folosind simulatorul M-SIM 2 [42] şi l-am extins variind mai mulţi parametri. În DSE-ul manual au fost variaţi doar 2 parametri. Noi, cu ajutorul lui FADSE, am variat 19 parametri (printre aceştia şi dimensiunea predictorului de valori SLVP propus de domnul Dr. Árpád Gellért). Aceşti parametrii generează un spaţiu de căutare de peste 2,5 milioane de miliarde de configuraţii posibile. Am folosit diferite tehnici de introducere a cunoştinţelor de domeniu: constrângeri, startarea procesului de DSE de la o populaţie cu indivizi consideraţi de noi buni, reguli fuzzy.

Următorul pas a fost să trecem la o arhitectură multi-core folosind simulatorul M-SIM3.

În toate experimentele am optimizat două obiective: CPI (clocks per instruction) şi energie. Ambele trebuie minimizate.

Ca şi benchmark-uri am utilizat suita SPEC 2000.

6.1 Optimizarea Arhitecturii Simulate de M-SIM 2

Scopul acestei cercetări este de a verifica diferite tehnici de introducere a cunoştinţelor de domeniu în algoritmii de DSE. Un alt deziderat a fost de a găsi configuraţii, ale procesorului Alpha simulat mai bune decât cele găsite manual de către domnul Dr. Árpád Gellért. Am dorit şi să demonstrăm că predictorul SLVP propus în [39] conduce la o scădere a energiei consumate de către sistem la acelaşi CPI.

Această cercetare a fost trimisă spre publicare la revista IET Computers & Digital Techniques [43].

6.1.1 Metodologie

6.1.1.1 Explorarea Manuală

Primul pas a fost să refacem DSE-ul manual realizat de domnul Dr. Árpád Gellért. Configuraţiile obţinute în urma acestei explorări au fost folosite în următoarele experimente ca puncte de start. Am variat dimensiunea cache-urilor.

6.1.1.2 Rulări Fără Informaţii Iniţiale

După DSE-ul manual am pornit FADSE de la o populaţie iniţială aleatoare. În plus am setat constrângeri pentru dimensiunile maxime si minime ale cache-urilor. După această rulare am observat că rezultatele nu sunt foarte bune, algoritmul nu explora suficient zona din spaţiul configuraţiilor care consuma puţină energie.

Am analizat rezultatele si am observat că limitele de jos ale cache-urilor erau prea mari si algoritmul nu caută în acea zonă. Am redus constrângerile impuse asupra dimensiunii minime a cache-urilor de nivel 1 si 2 si am repornit explorarea.

6.1.1.3 Rulare Pornind de la o Populație Inițială

În acest experiment am pornit de la o populație inițială care nu mai este complet aleatorie. O parte dintre indivizii populației inițiale sunt cei mai buni indivizi găsiți în explorarea manuală plus câteva din vecinătățile lor (cache-uri mai mari sau mai mici). Am pornit de la o populație considerată de noi bună cu scopul de a accelera procesul de explorare.

6.1.1.4 Rulări cu Reguli Fuzzy

Ultimele seturi de experimente au testat regulile fuzzy. Am propus câteva reguli fuzzy pe care le-am introdus în FADSE. Regulile propuse de noi sunt:

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*

Pentru dimensiunea cache-urilor am folosit parametri virtuali.

6.1.2 Rezultate

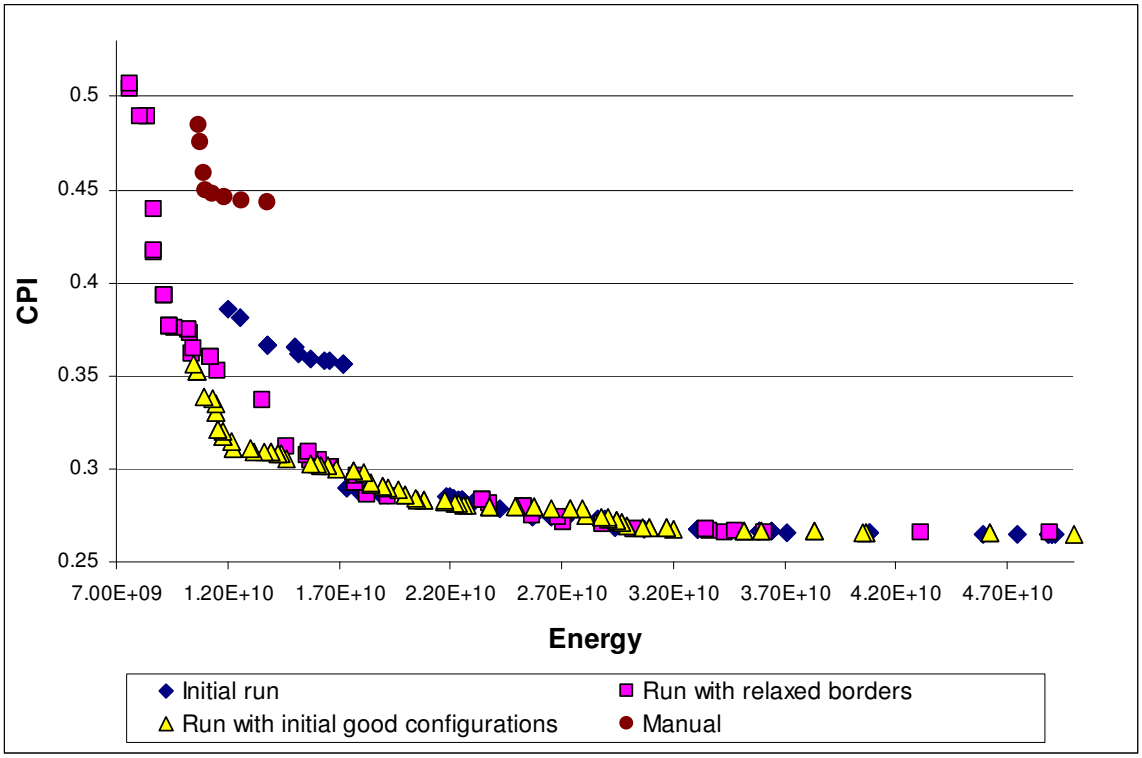

Figura 6.1-1 Comparație între fronturile Pareto aproximative găsite de primele trei rulări

In Figura 6.1-1 facem o comparație între primele trei rulări: cu prima versiune de constrângeri, cu constrângeri relaxate şi cu indivizi buni înserați în prima generație. Toate aceste rulări sunt comparate cu configurațiile obținute manual.

Rularea inițială (initial run în figură) cu constrângeri nerelaxate găseşte rezultate foarte bune din punct de vedere a vitezei de procesare dar nu reuşeşte să exploreze suficient zona unde energiile sunt mici. Există configurații obținute manual cu o energie consumată mai mică decât cele găsite automat. Am analizat rezultatele si am observat că în configurațiile obținute manual avem cache-uri ceva mai mici decât

cele găsite automat, care nu scădeau sub o anumită dimensiune. Am ajuns la concluzia că algoritmul tinde să "fugă" din zona graniţelor impuse de constrângeri. Cu aceste constrângeri zonele de graniţă nu erau atinse. Am relaxat limitele inferioare ale constrângerilor permiţând astfel cache-uri mult mai mici să apară în configuraţiile fezabile. Spaţiul de căutare s-a dublat (totuşi rămâne la 3% din spaţiul total de căutare). În această a doua rulare am obţinut rezultate foarte bune. Am depăşit configuraţiile obţinute manual pe ambele obiective. Indivizii sunt răspândiţi uniform pe suprafaţa Pareto. Recomandăm designerilor să relaxeze constrângerile aplicate chiar dacă configuraţiile de pe graniţe sunt nepractice. Această relaxare va permite algoritmului să exploreze mai bine zona apropiată de graniţe.

Am păstrat constrângerile stabilite la rularea anterioară, dar am pornit algoritmul de la o populaţie în care am inserat configuraţiile obţinute manual si câteva vecinătăţi (vecinătăţi ale dimensiunii cache-ului). Rezultatele au fost bune, în unele zone au depăşit rularea cu graniţe relaxate, dar totuşi nu explorează zona cu energii mici. Pentru a afla cauza acestei probleme am observat evoluţia frontului Pareto pe toată durata procesului de DSE. Am observat că aceste configuraţii iniţiale inserate în populaţie sunt foarte performante în comparaţie cu restul indivizilor produşi de algoritm. Ele vor supravieţui fiecărei generaţii si are loc o aglomerare în acea zonă. Problema cu aceste configuraţii este că nu sunt foarte diverse, ele diferă doar prin doi parametrii (numărul de seturi din cache-ul de nivel 1 si 2). Această lipsă de diversitate duce algoritmul intr-o minimă locală pe anumite zone ale frontului Pareto.

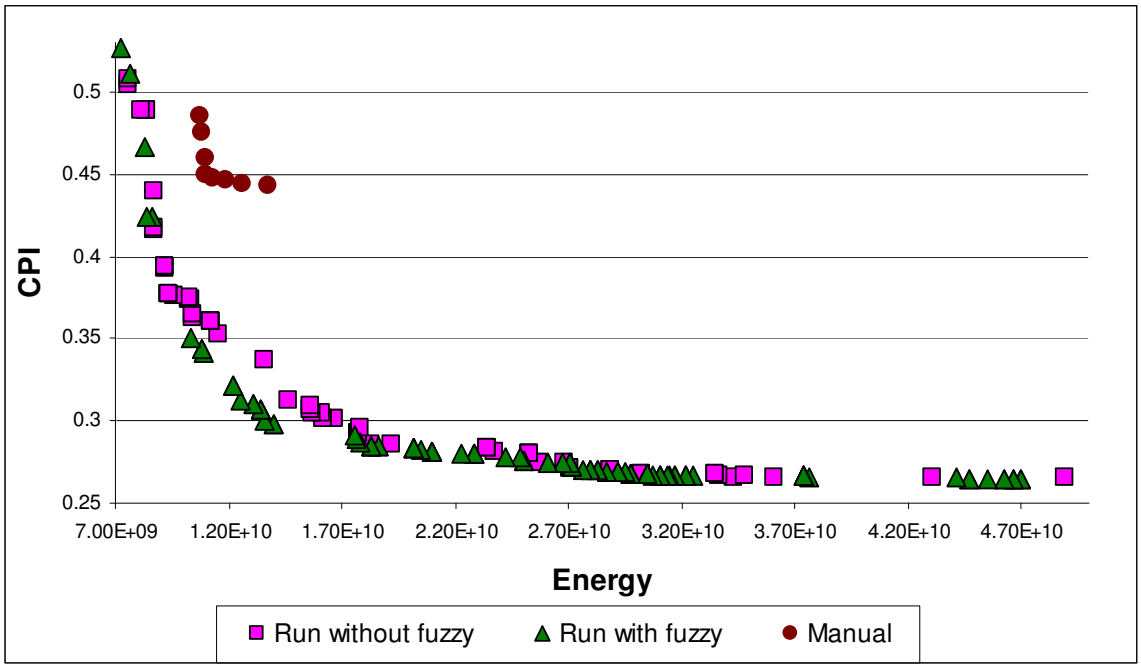

Figura 6.1-2 Comparaţie a fronturilor Pareto aproximative obţinute de rularea cu graniţe relaxate si a rulării cu reguli fuzzy

În ultimele rulări am testat regulile fuzzy cu ambii operatori de mutaţie propuşi de noi. în Figura 6.1-2 comparam rularea cu reguli fuzzy cu o probabilitate constantă de aplicare cu experimentul cu constrângeri relaxate. Se vede că rularea cu reguli fuzzy obţine rezultate mai bune. Rularea cu o distribuţie Gaussiană a probabilităţii de a aplica regulile fuzzy nu obţine rezultate atât de bune.

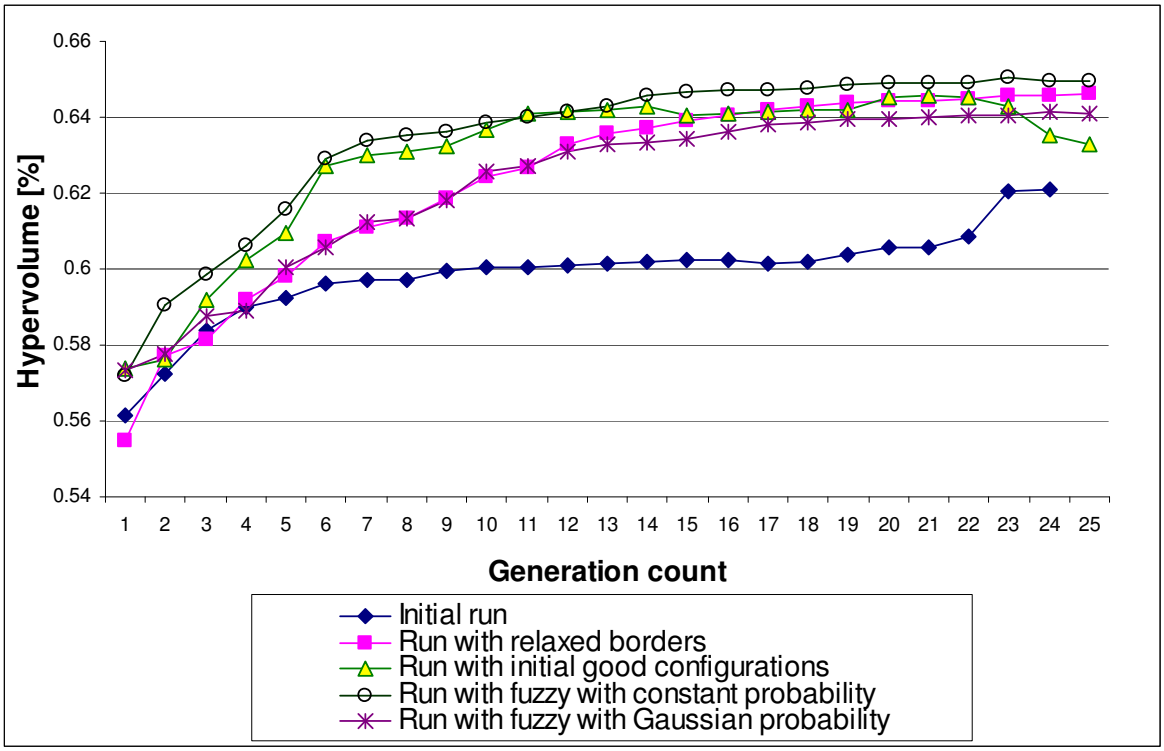

Figura 6.1-3 Comparație a hipervolumului obținut în toate experimentele

Ca şi o comparație finală am calculat hipervolumul obținut de toate rulările (vezi Figura 6.1-3). Aceasta ne dă o informație asupra vitezei de convergență a algoritmilor, dar si informații legate de calitatea soluțiilor obținute. Observăm o convergență rapidă a rulării cu informații date prin reguli fuzzy (probabilitate constantă) si a rulării care porneşte da la o populație de indivizi considerați buni. Ca şi calitate a soluțiilor rularea cu informații fuzzy obține cele mai bune rezultate.

Trecerea la o distribuție de probabilitate Gaussiană de a aplica regulile fuzzy duce la rezultate mai slabe. Am ajuns la concluzia că probabilitatea mare de a aplica regulile fuzzy duce la o pierdere în diversitate. Avem doar câteva reguli, cu puține intervale de membership asociate termenilor. Acest lucru înseamnă că toți indivizii vor fi mutați în intervale asemănătoare, rezultând indivizi similari. Când probabilitatea de aplicare a regulilor fuzzy scade operatorul de mutație nu mai este în stare să schimbe valorile parametrilor implicați pentru a ieşi din acest minim local. Am testat aceşti doi operatori de mutație (probabilitate constantă, distribuție Gaussiană de probabilitate) si pe GAP. Acolo situația a fost diferită. Aveam multe reguli si termenii utilizați în reguli aveau multe intervale de membership asociate. Distribuția Gaussiană de probabilitate a dat rezultate mai bune.

Ca şi concluzie finală putem spune că informația dată de designer poate ajuta procesul de DSE dar nu în toate cazurile. Trebuie avut grijă să nu se piardă diversitatea prin adăugarea acestei informații. Ajutorul dat se vede atât în calitatea soluțiilor cât si în viteza de convergență. O viteză mai mare de convergență duce la un timp mai redus necesar explorării. În cazul lui M-SIM 2 câştigul este de aproximativ 9 zile de simulare: rularea cu reguli fuzzy a ajuns la hipervolumul atins de rularea cu reguli relaxate în 19 zile fata de cele 24 necesare celeilalte rulări (durează 24 de ore pentru a evalua o generație cu 96 de clienți pe sistemul HPC de la Universitatea „Lucian Blaga").

Am analizat şi gradul de reutilizare. Am obținut o reutilizare de 12% ceea ce înseamnă o reducere cu 12% a timpului de evaluare.

În plus, prin experimentele făcute am demonstrat că predictorul selectiv de loaduri (SLVP) duce la o scădere a consumului de energie al arhitecturii simulate.

6.2 Optimizarea Arhitecturii Simulate de M-SIM 3

Următorul nostru obiectiv a fost să evaluăm o arhitectură multi-core. Am utilizat simulatorul M-SIM 3 pe care l-am configurat să ruleze cu două core-uri. Evaluarea unei generații durează 36 de ore pe 96 de core-uri Intel Xeon.

Obiectivele de optimizat au rămas energia şi CPI-ul.

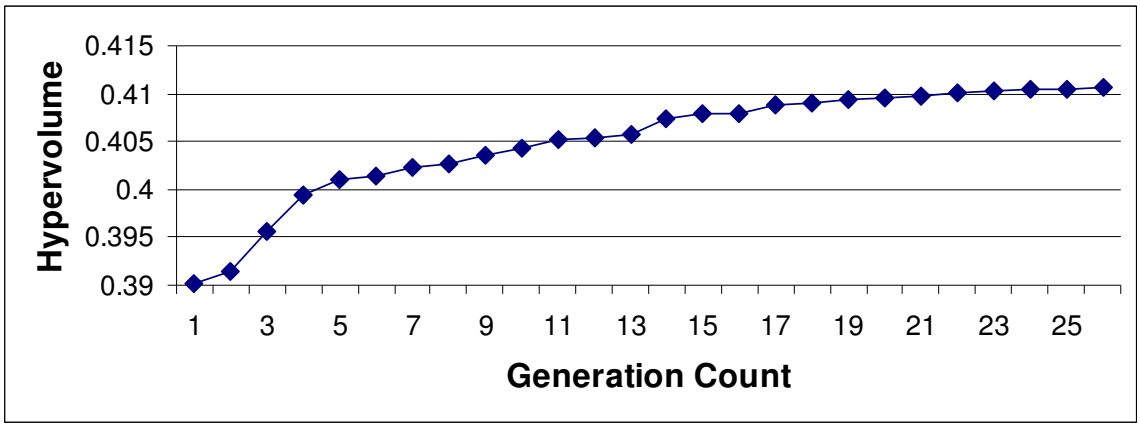

Figura 6.2-1 Hipervolumul obținut cu FADSE pe simulatorul M-SIM 3

Rezultatele obținute au fost foarte bune (vezi Figura 6.2-1). Cu această cercetare am demonstrat faptul că FADSE poate fi folosit cu succes pentru optimizarea de arhitecturi multi-core. În plus am demonstrat încă o dată faptul că FADSE este o unealtă robustă (rularea celor 26 de generații a durat mai mult de o lună).

6.3 Simulatoare Multi-core Analizate în Vederea Optimizării

Mai multe simulatoare multi-core au fost luate în considerare pentru optimizare folosind FADSE. Am analizat toate aceste simulatoare pentru a încerca să găsim punctele lor forte şi punctele lor slabe. Dintre simulatoarele analizate amintim: UNISIM [44], Multi2Sim, M5 [45], SESC [46], SCoPE si Graphite. Pentru o parte din ele am şi dezvoltat conectori pentru FADSE.

Marele dezavantaj al multor simulatoare este lipsa calculului de consum de putere. Acesta a şi fost motivul pentru care am utilizat M-SIM 3 în experimentele noastre.

Am integrat în simulatorul UNISIM noi protocoale de coerență, am scris benchmark-uri paralele pentru el, am dezvoltat o aplicație care permitea reconfigurarea simulatorului printr-o interfață grafică.

Am început integrarea librăriei McPAT [47] în simulatorul Multi2Sim. Am dezvoltat un conector pentru acest simulator.

Am implementat un conector cu FADSE şi pentru simulatorul M5.

SCOPE si Graphite sunt două simulatoare cu care lucrăm în acest moment şi sperăm să le integrăm cât mai curând în FADSE.

“An ounce of action is worth a ton of theory.”

Friedrich Engels

7 Optimizări Multi-Obiectiv a Arhitecturilor System on Chip

În acest capitol am integrat simulatorul UniMap [48] dezvoltat de Radu Ciprian cu FADSE şi ne propunem să realizăm un DSE pe SoC-uri (System on Chip). Scopul cercetării este de a găsi o arhitectură SoC cât mai bună pentru o anumită aplicație. Ne ocupam şi de problema mapării aplicațiilor pe rețeaua de comunicație NoC (Network on Chip) integrată în SoC. Pentru aceasta propunem un workflow de DSE fezabil care serializează procesul de mapare şi cel de explorare pentru a evita complexitatea enormă a acestei cautări. Obiectivele optimizate în cadrul acestei cercetări sunt: consumul de energie, aria de integrare şi timpul de rulare al aplicației.

7.1 Procesul de DSE

Prima fază a procesului de DSE constă în maparea aplicațiilor pe rețeaua NoC folosind un model analitic. Acest model analitic utilizează ca informație de fitness doar consumul de energie (estimat analitic) [49].

Am folosit mai mulți algoritmi de mapare, unii bine cunoscuți, alții propuşi de către Radu Ciprian: Simulated Annealing, Branch and Bound, Optimized Simulated Annealing, Elitist Genetic Algorithm si Elitist Evolutionary Strategy [50]. Toate mapările sunt stocate într-o bază de date. Din această bază de date am selectat cele mai bune mapari găsite analitic: primele 10 mapări pentru benchmark-ul telecom, iar pentru restul benchmark-urilor (MPEG, H.264, VOPD), câte o mapare, datorită timpului mare necesar unei rulări.

FADSE primeşte aceste mapări (vezi Figura 7.1-1) şi începe procesul de DSE prin care caută parametrii arhitecturii. Pe lângă parametrii rețelei (frecvența de operare, dimensiunea de buffere) FADSE selectează şi tipul de core asignat fiecărui nod alegând dintr-o librărie de core-uri IP. După ce a construit o instanță de arhitectură, FADSE porneşte simulatorul inclus în UniMap şi obține valorile obiectivelor (timpul de execuție al aplicației, energia consumată de sistem şi aria ocupată de acesta). Procesul este reluat şi algoritmul inclus în FADSE învață din aceste rezultate.

A fost necesar să serializăm procesul de căutare a unei mapări cvasi optimale şi procesul de găsire a unor parametrii arhitecturali buni din motive de complexitate. Mii de mapării sunt analizate pentru a găsi o mapare bună. Folosirea simulatorului pentru această sarcină ar duce la un consum de timp exagerat. Prin selectarea a doar câtorva mapări şi realizarea căutării a unui set de parametrii optimal, doar pe acestea, ar reduce timpul necesar explorării.

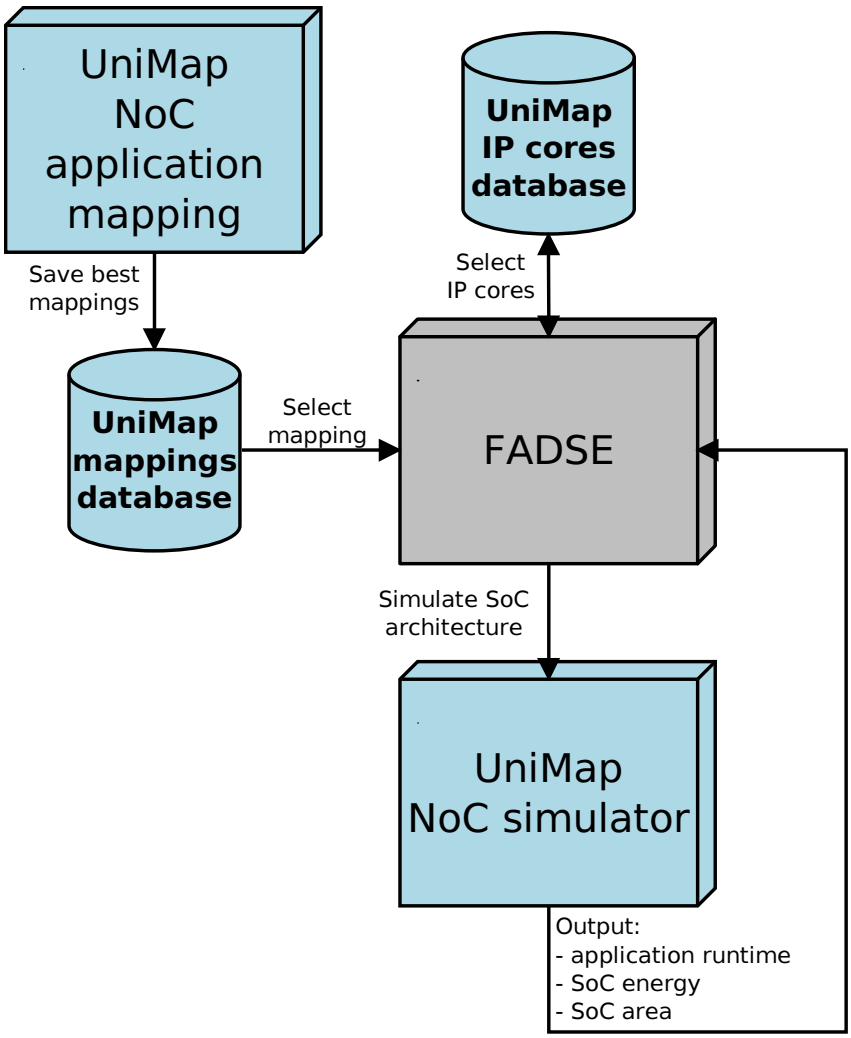

Figura 7.1-1 Workflow-ul propus de noi pentru DSE pe SoC

7.2 Rezultate

In acest capitol prezentăm câteva rezultate preliminare obținute în acest experiment. Am optimizat arhitectura SoC pe patru benchmark-uri: telecom, MPEG-4, H.264 si VOPD. în funcție de benchmark (număr de core-uri, tipuri de core-uri permise pentru executarea fiecărui task) dimensiunea spațiului de căutare variază de la 10^{22} până la 10^{34}.

Algoritmii testați sunt: NSGA-II, SPEA2 (genetici) si OMOPSO, SMPSO, de tip particle swarm optimization (PSO).

Vom prezenta doar rezultatele obținute pe benchmark-ul telecom. Concluziile se aplică si celorlalte benchmark-uri.

In Figura 7.2-1 am comparat valoarea medie a hipervolumului obținută de cei 4 algoritmi pe durata a 50 de generații. Fiecare algoritm a fost rulat pe cele mai bune 10 mapări găsite în mod analitic. Se poate observa că algoritmii genetici obțin cele mai bune rezultate. Algoritmii PSO au o viteză de convergență mai mare dar sunt depăşiți în calitatea soluțiilor de către algoritmii genetici după 8-9 generații.

Am realizat şi o comparație folosind metrica coverage între cei 4 algoritmi (doi câte doi) şi am concluzionat că SPEA2 obține cele mai bune rezultate.

În Figura 7.2-2 am reprezentat frontul Pareto obținut prin selectarea indivizilor nedominati după combinarea tuturor rezultatelor obținute de către toți algoritmii.

Mapările au fost obținute analitic şi sunt primele 10 cele mai bune (maparea 1 este cea mai bună, maparea 2 este pe locul doi şi aşa mai departe). Este interesant de văzut că obținem puncte din toate mapările. Am obținut cel mai bun consum de energie cu a şasea mapare. Cea mai mică arie a fost dată de mapările a treia şi a cincia. Cu exact aceeaşi arie, a treia mapare are o energie mai bună, în timp ce a cincia are un timp de rulare al aplicației mai bun. În sfârşit, cel mai scăzut timp de rulare al aplicației a fost găsit pe un design SoC corespunzător tot mapării a şasea.

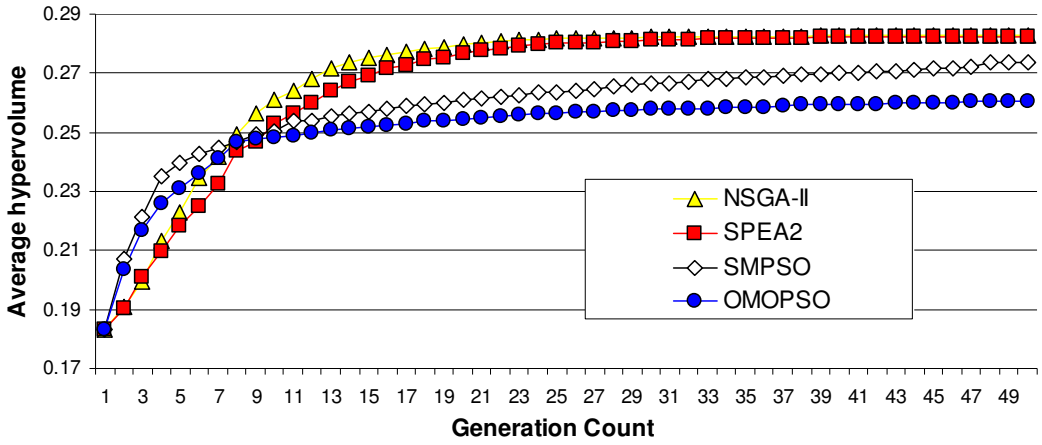

Figura 7.2-1 Hipervolumul mediu obținut de cei 4 algoritmi după cele zece rulări

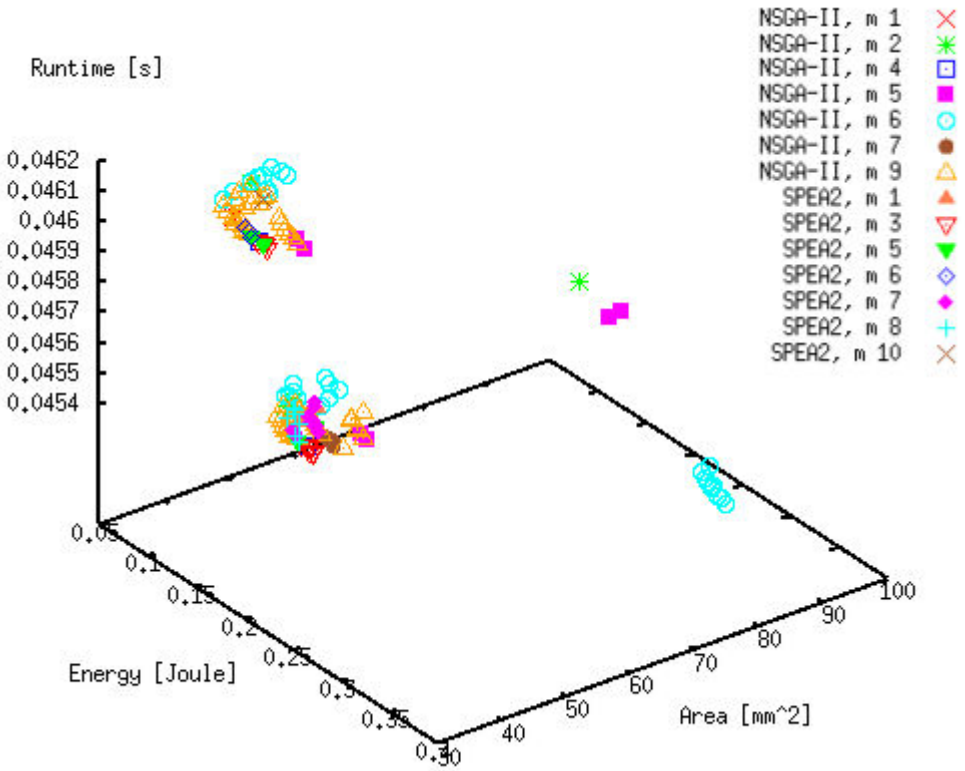

Figura 7.2-2 Frontul Pareto obținut de SPEA2, pentru benchmark-ul telecom

Nu am obținut cea mai bună energie cu prima cea mai bună mapare, care analitic ne-a dat cea mai scăzută energie de comunicare NoC. Acest lucru poate fi din cauza mai multor factori. În primul rând, noi am estimat analitic numai energia de

comunicare NoC. Cu această abordare am calculat întreaga energie SoC (energia core-urilor este de asemenea inclusă). În al doilea rând, modelul analitic nu poate capta efectele dinamice ale rețelei (congestiile din rețea). În al treilea rând, FADSE nu efectuează, în mod evident, o căutare exhaustivă. Este posibil ca noi să putem obține rezultate mai bune pe energie, cu prima mapare decât cu cea de a opta. Acest lucru demonstrează necesitatea de a efectua o mai bună explorare a spațiului de proiectare. Utilizarea cunoştinţelor de domeniu pentru a limita spațiul de căutare şi de aplicare a regulilor fuzzy sunt două abordări care ar putea îmbunătăți tehnica DSE.

O altă observație pe care o putem face legată de această figură este că obținem soluții nedominate doar cu algoritmii genetici.

Concluzia după acest experiment si având si rezultatele de la experimentul pe GAP unde am comparat algoritmii NSGA-II, SPEA2 si SMPSO este că nu există un algoritm cel mai bun. Fiecare algoritm obține rezultate mai bune pe anumite probleme. Totuşi ce putem concluziona este că toți algoritmii vor genera în final rezultate bune.

7.3 Îmbunătățiri ale Sistemului Many-Core MANJAC

MANJAC [53] este un sistem cu 64 de procesoare multi-core, multi-threaded capabile să execute cod nativ JAVA. Aceste procesoare sunt aşezate întrun mesh 8x8. Fiecare procesor conține 6 core-uri şi fiecare core este capabil să execute 4 thread-uri (SMT).

În cadrul perioadei de mobilitate efectuată la Universitatea din Augsburg am lucrat cu acest sistem cu scopul de a porta un middleware scris în JAVA pe această arhitectură. Pentru aceasta a trebuit să înlocuiesc librăria de comunicare folosită (JXTA - http://en.wikipedia.org/wiki/JXTA) cu o implementare mai simplă care să îndeplinească în mare parte aceleaşi funcții.

În plus am implementat şi noi metode de inițializare pentru sistemul MANJAC, capabile să evite diferite probleme care pot apărea în rețea. În acest scop am dezvoltat şi o aplicație GUI capabilă să monitorizeze procesul de inițializare al sistemului.

Mai multe detalii pot fi găsite în raportul nostru “Introduction to the MANJAC system” [54].

8 Concluzii şi Dezvoltări Ulterioare

Această teza are următoarele contribuții:

- Am propus clasificarea algoritmilor în două clase: algoritmi evolutivi şi algoritmi bio-inspirați. Această clasificare ne-a ajutat în timpul experimentelor. În rezultatele obținute în Capitolul 7 am putut observa că performanțele algoritmilor depind de clasa din care fac parte: algoritmii evolutivi au obținut rezultate asemănătoare iar cei bio-inspirați de asemenea s-au grupat obținând rezultate apropiate.
- Am comparat doi algoritmi evolutivi simpli (SEMO şi FEMO) pe două probleme sintetice (LOTZ şi DTLZ1). Folosind diferite metrici de performanță (coverage şi error ratio) am demonstrat că FEMO obține rezultate mai bune. Pe problema LOTZ ambii algoritmi găsesc soluțiile optimal într-un timp relativ scurt (1%-2% din timpul necesar unei evaluări exhaustive).Totuşi, nici unul dintre cei doi algoritmi nu reuşeşte să găsească frontul Pareto optim (true Pareto front) când DTLZ1 a fost folosită ca problemă de test.
- Am analizat doi algoritmi genetici foarte cunoscuți (NSGA-II şi SPEA2). În urma comparării am ajuns la concluzia că SPEA2 va conține mai mulți indivizi identici în populația finală. Acest lucru a fost confirmat şi de rezultatele experimentale obținute atât pe simulatorul procesorului GAP (împreună cu optimizatorul lui de cod) cât şi pe simulatorul de NoC-uri UniMap.
- Am dezvoltat o unealtă pentru DSE denumită FADSE. Include mulți algoritmi state-of-the-art prin integrarea cu librăria jMetal. De asemenea am implementat mai multe metrici (în special cele care nu necesită cunoaşterea lui true Pareto front).
- Una dintre cele mai importante facilități ale lui FADSE, care nu apare în nici una din uneltele publice dedicate DSE-ului pentru arhitecturi de calculatoare pe care noi le-am analizat, este evaluarea distribuită a configurațiilor. Am profitat de paralelismul intrinsec oferit de majoritatea algoritmilor euristici de căutare. Am dezvoltat versiuni paralele pentru următorii algoritmi: AbBYS, Densea, FastPGA, IBEA, NSGA-II, OMOPSO, PESA2, SMPSO şi SPEA2.
- FADSE permite o flexibilitate crescută în alocarea resurselor. Numărul de clienți care simulează poate fi crescut/scăzut în mod dinamic în timp ce explorarea rulează. Acest lucru îi permite utilizatorului să folosească cât mai bine resursele computaționale de care dispune.
- Am testat FADSE pe diverse sisteme: HPC-ul de la Universitatea din Sibiu (120 de clienți cu procesoare Intel Xeon), un cluster cu procesoare IBM Cell, pe grid-ul de la Universitatea Politehnică Bucureşti (100 de clienți atât pe procesoare Opteron cât şi pe Nehalem), o maşină virtuală bazată pe Windows cu 32 de procesoare situată la Universitatea din Augsburg.
- Pentru a accelera şi mai mult procesul de DSE am integrat FADSE cu un sistem de baze de date (MySQL). Acesta ne permite să reutilizăm rezultate obținute anterior. Am ajuns la grade de reutilizare de peste 67% doar dintr-o singură rulare (algoritmii tind să producă din nou aceiaşi indivizi), având în vedere că am reutilizat şi din explorări anterioare, acest număr ar putea fi mai mare. Această reutilizare reduce considerabil timpul necesar explorării.

- Explorările pot dura zile, săptămâni sau chiar luni. Pentru asta FADSE trebuie să fie o aplicație robustă. Am implementat diferite tehnici care pot detecta şi evita diferite probleme care pot apărea: clienții nu mai răspund, rețeaua de comunicare nu mai funcționează. În aceste situații FADSE retrimite indivizi de evaluat la alți clienți.
- Pot apărea probleme mult mai grave: serverul nu mai funcționează sau alimentarea de la rețeaua electrică se întrerupe. Pentru a evita repornirea procesului de DSE am implementat un mecanism de checkpointing. Acesta salvează starea explorării la anumite momente de timp. În caz că este nevoie de repornirea simulării se poate continua de la un asemenea checkpoint, timpul pierdut fiind minim. Mecanismul de checkpointing ne permite de asemenea să pornim FADSE de la o populație inițială definită de utilizator. Am folosit această opțiune în toate capitolele din teză. De exemplu în comparările de algoritmi când ne dorim ca toți algoritmii să pornească de la aceeaşi configurație inițială pentru a face o comparație cât mai corectă. O altă situație este cea în care am introdus în populația inițială configurații găsite manual, considerate bune, pentru a accelera procesul de DSE.
- FADSE este orientat spre optimizarea sistemelor de calcul. Am dezvoltat o interfață uşor de folosit prin care FADSE se poate conecta la orice simulator, pentru un sistem de calcul, existent. Există mai multe clase ajutătoare în FADSE care îi ascund dezvoltatorului problemele legate de integrarea cu baze de date, tratarea erorilor şi mesajelor de la server. Acestea îi permit dezvoltatorului să scrie un conector pentru simulatorul dorit întrun timp relativ scurt.
- Cu ajutorul studenților am dezvoltat mai mulți conectori pentru diferite simulatoare de sisteme de calcul: GAP, GAPtimize, M-SIM 2, M-SIM 3, UniMap, M5 şi Multi2Sim.
- Pornind de la interfața de configurare propusă de M3Explorer am dezvoltat propria interfață XML. Din aceasta interfață se pot configura uşor atât parametrii lui FADSE cât şi cei ai arhitecturii. Parametrii dați simulatorului pot fi de tip întreg (progresii aritmetice şi geometrice cu diferite rații), liste de stringuri, etc. Pe lângă acesta utilizatorul mai poate specifica constrângeri între parametri, ierarhii de parametri, obiectivele care trebuiesc optimizate. Tot din XML se configurează calea către simulator, fişierele de ieşire, conexiunea la baza de date, lista de benchmark-uri folosite pentru simulări, etc.
- Am analizat mai multe metrici pentru evaluarea performanțelor algoritmilor de căutare. Ne-am focalizat în special pe cele care nu necesită cunoaşterea frontului Pareto optim: coverage, hipervolum, two set difference hypervolume, pe care le-am şi implementat în FADSE.
- Am inclus în FADSE mai multe metode care să-i permită utilizatorului să-şi exprime cunoştințele de domeniu: constrângeri între parametri, ierarhii de parametri, reguli fuzzy.
- Constrângeri între parametri au fost propuse şi folosite şi de alte utilitare de DSE. Am implementat şi noi în FADSE o interfață prin care utilizatorul poate defini astfel de constrângeri. Algoritmii abordează tehnica propusă de Deb de tratare a indivizilor nefezabili (indivizi care nu respectă constrângerile). Am observat că în momentul în care constrângerile sunt aplicate există tendință de a apărea mulți indivizi nefezabili în populație (depinde de constrângeri şi cât reduc ele dimensiunea spațiului). Existența multor indivizi nefezabili în

populație limitează numărul de indivizi buni pe care algoritmul îi poate alege, ceea ce duce la o convergență mai lentă. Am schimbat algoritmii de căutare şi, dacă utilizatorul doreşte, pot fi forțați să producă un număr minim de indivizi fezabili până să treacă la următoarea generație. Această modificare a îmbunătățit rezultatele (am folosit mai departe această tehnică în majoritatea experimentelor rulate).

- De multe ori există situații când anumiți parametri sunt activi doar în unele cazuri. De exemplu, dacă configurația de procesor foloseşte un branch predictor de un anumit tip, atunci doar parametrii acelui predictor trebuie să fie activi nu şi al altui tip. Am analizat diferite situații care pot apărea şi am implementat un mecanism prin care aceste relații între parametri (ierarhii de parametri) pot fi exprimate uşor. Informația dată de utilizator este transmisă algoritmului care trebuie să țină cont de ea. Pentru aceasta am propus noi operatori de mutație şi crossover. Aceasta este o cercetare aflată încă la început.
- O contribuție importantă a acestei teze este includerea în algoritmii de căutare a informațiilor date de utilizator sub forma de reguli fuzzy. Din ceea ce ştim suntem primii care au propus utilizarea de reguli fuzzy ca metodă apriori de introducere a informației în algoritmii de DSE pentru arhitecturi de calculatoare. Descrierea acestor reguli fuzzy se face uşor printr-un limbaj apropiat celui natural. Ele permit designer-ului să îşi exprime cunoştințele legate de domeniu. Pentru a permite utilizarea de reguli fuzzy în FADSE am integrat unealta dezvoltată de noi cu librăria jFuzzyLogic. Aceasta librărie permite descrierea regulilor întrun limbaj standard (FCL – Fuzzy Control Language), în plus include mai multe sisteme de inferență. Noi am folosit sistemul de inferență propus de Mamdani, unul dintre cele mai folosite şi citate în literatură. Şi în acest caz informația obținută din regulile fuzzy trebuie să fie luată în calcul de algoritmul de DSE. Pentru aceasta am propus şi implementat doi noi algoritmi de mutație. Ambii sunt bazați pe mutația de tip bit flip pe numere întregi. Diferența între cei doi constă în modul în care se calculează probabilitatea de a utiliza informația obținută în urma defuzificării. Primul operator propus foloseşte o probabilitate constantă de a aplica informația dată de regulile descrise de către utilizator. Această probabilitate (fuzzy probability) este egală cu probabilitatea de mutație. Dacă parametrul curent este ales pentru o „mutație fuzzy" atunci valoarea lui va fi egală cu centrul de greutate al funcției de membership obținută în urma aplicării procesului de inferență asupra regulilor fuzzy. Al doilea operator propus foloseşte o probabilitate de mutație care nu este constantă în timpul rulării algoritmului de DSE. Am folosit o distribuție Gaussiană a probabilității cu scopul de a avea o probabilitate mare de aplicare a informației date de regulile fuzzy pentru primele generații ale algoritmului. Pe măsură ce algoritmul rulează aceasta probabilitate scade şi îl lăsăm liber să caute. Totuşi, chiar şi pentru primele generații nu am lăsat o probabilitate mai mare de 80% de a aplica regulile fuzzy. Această probabilitate scade până când devine egală cu probabilitatea de mutație. Tot în această ultimă metodă folosim şi informații legate de membership-ul valorii centrului de greutate obținut în urma defuzificării. Valoarea membership-ului este folosită ca o confidență în valoarea obținută în urma defuzificării.

- Am observat că în multe situații designer-ul foloseşte termeni care îmbină mai mulți parametri, de exemplu dimensiunea cache-ului de nivel 1, care este dată de parametrii: numărul de linii din cache, dimensiunea blocului, gradul de asociativitate. Pentru a trata astfel de cazuri am implementat „parametri virtuali". Aceştia pot fi definiți ca o combinație de mai mulți operatori şi folosiți apoi în interiorul regulilor fuzzy ca parametri de intrare.
- Am propus un defuzificator aleatoriu care îi permite utilizatorului să specifice valoarea minimă a membership-ului valorilor luate în considerare de către defuzificator. Am folosit acest defuzificator când forma funcțiilor de membership este (aproape) rectangulară.
- Am propus o metodă de calculare a complexității hardware a procesorului GAP.
- Am realizat un DSE automat al procesorului GAP cu rezultate foarte bune. Am demonstrat că FADSE poate găsi rezultate mai bune decât un designer uman. În explorarea procesorului GAP am găsit configurații cu acelaşi CPI dar cu o complexitate hardware de două ori mai mică. Am demonstrat că designerii pot fi subiectivi şi să aleagă parametrii în funcție de pre-concepții greşite, neobservând legături subtile între parametri.
- Am realizat optimizări cu un singur obiectiv utilizând FADSE pe optimizatorul de cod GAPtimize. Rezultatele au fost bune şi ne-au determinat să mergem mai departe şi să optimizăm în acelaşi timp parametrii hardware ai lui GAP cât şi parametrii software ai lui GAPtimize. Şi în acest experiment rezultatele au fost foarte bune şi am demonstrat că FADSE poate fi folosit pentru optimizări hardware şi software în paralel.
- Am realizat o comparație a trei algoritmi euristici de DSE: doi genetici (NSGA-II si SPEA2) şi unul de particle swarm optimization (SMPSO). I-am comparat atât pe DSE-uri făcute pe GAP cât şi pe GAP împreună cu GAPtimize. Am arătat cu această ocazie că SMPSO converge cel mai repede şi obține şi cele mai bune rezultate. Nu putem recomanda unul dintre cei doi algoritmi genetici. NSGA-II a obținut rezultate mai bune în prima explorare (GAP) pe când SPEA2 a avut rezultate ceva mai bune în al doilea experiment (GAP+GAPtimize). Totuşi, după analiza fronturilor Pareto aproximative obținute de cei trei algoritmi, am ajuns la concluzia că diferențele între indivizii (configurațiile) obținuți nu sunt foarte mari. Toți algoritmii găsesc soluții foarte bune. SPEA2 tinde să nu aibă o distribuție a indivizilor la fel de uniformă ca a celorlalți algoritmi de-a lungul suprafeței Pareto. Tot după această analiză am concluzionat că metrica coverage ar putea induce în eroare un designer, arătând o diferență mare între algoritmi când de fapt în realitate ea nu există. Am propus utilizarea de metrici care să folosească un treshold pentru stabilirea relației de dominantă (e-dominance). Acest treshold ar permite designer-ului să stabilească de la ce distanță unul de altul indivizii sunt dominați.
- Am demonstrat că integrarea cu o bază de date poate duce la scăderi semnificative ale timpului necesar procesului de DSE. În experimentele pe GAP am obținut o reducere de peste 60% a timpului necesar explorării doar datorită reutilizării rezultatelor. Pe lângă mecanismul de reutilizare am utilizat şi evaluarea distribuită care duce la o scădere a timpului necesar unui DSE aproape liniar cu numărul de clienți (procesoare) disponibili.

- Am propus o metodă de generare a regulilor fuzzy din rezultatele obținute în urma explorărilor anterioare. Am folosit metode de învățare automată pentru a obține arbori de decizie din care am extras automat reguli pe care le-am integrat apoi în algoritmii de căutare prin intermediul mecanismului de reguli fuzzy puse la dispoziție de FADSE. Am arătat că aceste reguli îmbunătățesc rezultatele obținute şi cresc viteza de convergență a algoritmilor de căutare.
- Am testat parametrii ierarhici cu rezultate promițătoare folosind mai multe optimizări de cod din GAPtimize. Aceste optimizări au fiecare parametrii lor şi pot fi activate/dezactivate în timpul explorării.
- În capitolul 6 din teză am realizat un DSE pe procesorul Alpha. Am continuat munca făcută de domnul Dr. Árpád Gellért în teza sa de doctorat. Dânsul a realizat o explorare manuală a procesorului Alpha utilizând simulatorul M-SIM 2 în care a integrat şi un predictor de valori selectiv. Explorarea manuală a fost făcută variind doar doi parametri. Noi am decis să creştem numărul de parametri variați la 19, mărind astfel considerabil spațiul configurațiilor posibile. Căutarea manuală întrun spațiu atât de mare devine foarte dificilă şi atunci am folosit FADSE. În acest experiment am folosit mai multe tehnici de introducere a cunoştințelor de domeniu: constrângeri, pornirea de la configurații inițiale bune, reguli fuzzy (atât cu probabilitate constantă cât şi cu o distribuție Gaussiană a probabilității de aplicare a regulilor fuzzy).
- Am arătat că prin adăugarea de cunoştințe de domeniu în algoritmul de căutare rezultatele devin mai bune şi în acelaşi timp creştem viteza de convergență. Totuşi adăugarea de informații poate duce şi la rezultate ceva mai slabe dacă adăugarea lor duce la o pierdere a diversității. Am observat atât pe GAP cât şi pe M-SIM că utilizarea unei informații dată din exterior poate duce la pierderea diversității dacă informația nu este ea însăşi diversă. Cu alte cuvinte: am simulat cu M-SIM cu puține reguli, în plus termenii folosiți în reguli aveau puține funcții de membership asociate. Aplicarea forțată a acestor reguli pentru primele generații a dus la o pierdere a diversității. La GAP unde regulile au fost mai multe şi mai diverse, afectau mai mulți parametrii şi funcțiile de membership asociate termenilor erau mai multe, aplicarea cu o distribuție de probabilitate Gaussiană a regulilor a dus la rezultate mai bune. În alt experiment am introdus configurații considerate de noi bune în populația inițială. Am observat o creştere a vitezei de convergență dar spre finalul rulării s-a putut observa o reducere a diversității în populație şi pe anumite zone a suprafeței Pareto o imposibilitate a algoritmului de a găsi configurații, pe care în alte cazuri le-am găsit. Problema a fost că acele configurații inițiale erau foarte asemănătoare, difereau doar prin doi parametrii.
- Am integrat FADSE cu un simulator multi-core (M-SIM 3) pe care am şi făcut DSE. Cu această ocazie am demonstrat că FADSE se poate folosi şi pentru optimizarea de arhitecturi multi-core. În plus, acest experiment a fost un test pentru robustețea utilitarului dezvoltat de noi, simulările rulând mai mult de o lună de zile.
- Am realizat o analiză a simulatoarelor multi-core disponibile gratuit. Am îmbunătățit unele dintre ele cu noi protocoale de coerență, integrare de calcul de consum de putere dinamică, etc.
- Am integrat FADSE cu UniMap, un simulator SoC dezvoltat de Ciprian Radu în teza sa de doctorat. Împreună am realizat un DSE pe SoC-uri. Tot în acest experiment am comparat patru algoritmi de DSE: doi genetici (NSGA-II si

SPEA2) şi doi algoritmi bio-inspirați de tip particle swarm optimization (OMOPSO si SMPSO). În acest caz, în contrast cu experimentul făcut pe GAP şi GAPtimize, algoritmii genetici au avut rezultate mai bune, pe toate benchmark-urile pe care am rulat. Totuşi algoritmii de tip PSO au o viteză de convergență mai mare.

Pe viitor ne propunem să introducem o rețea neurală care să învețe din rezultatele produse de FADSE în timp ce procesul de DSE rulează. Când confidența în rețeaua neurală este suficient de mare (considerăm că ea a învățat) vom da indivizi aleatori la intrările rețelei şi ea va încerca să predicționeze ieşirile (obiectivele). Indivizii promițători vor fi injectați în populația copiilor din generația curentă.

Ne propunem să găsim noi metode de a utiliza informația dată de regulile fuzzy şi în alți operatori genetici.

9 Referințe

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

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

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

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

[5] 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.

[6] 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," în *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, 2002, pp. 44-53.

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

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

[9] 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.

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

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

[12] 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," în *Proceedings of the IEEE Symposium Series on Computational Intelligence*, 2009, pp. 66–73.

[13] 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.

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

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

[16] G. Palermo, C. Silvano, and V. Zaccaria, "Discrete Particle Swarm Optimization for Multi-objective Design Space Exploration," în *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*, 2008, pp. 641-644.

[17] 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.

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

[19] S. Uhrig, B. Shehan, R. Jahr, and T. Ungerer, "A Two-dimensional Superscalar Processor Architecture," în *The First International Conference on Future Computational Technologies and Applications, Athens, Greece*, 2009.

[20] C. Silvano, W. Fornaciari, and E. Villar, *Multi-Objective Design Space Exploration of Multiprocessor SOC Architectures: The Multicube Approach*. Springer, 2011.

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

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

[23] D. T. Pham and M. Castellani, "Action aggregation and defuzzification în 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.

[24] "IEC 61131-7 standard." [Online]. Available: http://www.fuzzytech.com/binaries/ieccd1.pdf.

[25] "International Electrotechnical Commission." [Online]. Available: http://www.iec.ch/.

[26] P. Cingolani, "jFuzzyLogic." [Online]. Available: http://jfuzzylogic.sourceforge.net/html/index.html.

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

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

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

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

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

[32] R. Jahr, T. Ungerer, **H. Calborean**, and L. Vintan, "Automatic Multi-Objective Optimization of Parameters for Hardware and Code Optimizations," în

Proceedings of the 2011 International Conference on High Performance Computing & Simulation (HPCS 2011), 2011, pp. 308 – 316.

[33] **H. Calborean**, R. Jahr, T. Ungerer, and L. Vintan, "Optimizing a Superscalar System using Multi-objective Design Space Exploration," în *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.

[34] 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.

[35] 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.

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

[37] 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," în *wwc*, 2001, pp. 3–14.

[38] **H. Calborean**, R. Jahr, T. Ungerer, and L. Vintan, "Optimizing a Superscalar System using Multi-objective Design Space Exploration," în *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.

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

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

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

[42] 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.

[43] 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).*

[44] 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.

[45] 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.

[46] J. Renau et al., *SESC simulator*. 2005.

[47] 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," în *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 469–480.

[48] C. Radu and L. Vințan, "UNIMAP: UNIFIED FRAMEWORK FOR NETWORK-ON-CHIP APPLICATION MAPPING RESEARCH," *Acta Universitatis Cibiniensis â€“ Technical Series*, May 2011.

[49] 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.

[50] C. Radu, "Optimized Algorithms for Network-on-Chip Application Mapping," PhD thesis, "Lucian Blaga" University of Sibiu, Romania, Sibiu, Romania, 2011.

[51] "The Embedded System Synthesis Benchmarks Suite (E3S) website." [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s/.

[52] 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," în *Proceedings of the Conference on Design, Automation and Test în Europe*, 3001 Leuven, Belgium, Belgium, 2009, pp. 423–428.

[53] S. Uhrig, "The MANy JAva Core processor (MANJAC)," în *HPCS*, 2010, p. 188.

[54] **H. Calborean**, "Introduction to the MANJAC system," Computer Science Department, "Lucian Blaga" University of Sibiu, Sibiu, Romania, 4, 2011.