Framework for Automatic Design Space Exploration of Computer Systems

Horia CALBOREAN, Lucian VINTAN

"Lucian Blaga" University of Sibiu, Romania, Engineering Faculty, Emil Cioran st. no. 4, 550025 Sibiu, Phone +40-269-217928, Fax +40-269-212716, e-mail {horia.calborean, lucian.vintan}@ulbsibiu.ro

Abstract: Recent years have shown an increasing complexity of the computer architectures. These led to an increased number of parameters that can be configured. To explore this huge design space system simulators are used. Since the traditional manual exploration will not scale as the complexity grows automatic tools are needed. A framework for automatic design space exploration called FADSE has been developed. We have already proved in some of our previous work that FADSE is able to discover good design points using evolutionary algorithms like: NSGA-II, SPEA2 and SMPSO on the GAP simulator. An overview of our prior work is presented in this article and also an analysis of the reuse of previously simulated individuals.

Key words: Automatic design space exploration, evolutionary and bio-inspired algorithms, particle swarm optimization, superscalar microarchitecture, simulation

1. Introduction

The complexity of the computer system has risen in the recent years. The number of transistors that can be integrated in a die has risen to over 2.9 billion[1] at the time this article was written. Multiple cores and complex structures are used to build these microprocessors. A large number of parameters have to be set to obtain a near optimal configuration. But with the high number of parameters the design space becomes enormous Vintan [1]. For example with 50 parameters each of them having 8 possible values the number of possible configurations is 2^{150}. On this huge design space an exhaustive search is impossible because evaluating only one single point might require several hours of simulation. Only a small fraction of the entire space can be evaluated to meet the time to market constraint.

With the increased number of parameters, the difficulty to predict the relations between them, the influence of a parameter on the architecture performance becomes increasingly harder to predict. The traditional method where a computer architect manually optimizes the architecture has become infeasible. New methods and tools are needed to help the designer and to speed up de search process. The task is further more hardened by the fact that the design space exploration process (DSE) is no longer a single objective search. The architect is required to optimize at the same time multiple objectives: speed (IPC), power consumption, area integration.

Tools that automate the design space exploration process and help the designer make better decisions are needed. We propose a tool developed by us, called FADSE (Framework for Automatic Design Space Exploration). Our tool contains many state of the art multi objective evolutionary algorithms which can automatically search through the design space and find quasi-optimal solutions.

The article is structured as follows: in section 2, basic concepts about design space exploration are introduced and a short presentation of the used metrics is performed. More details about our tool architecture and its functionality are offered in section 3. Section 4 describes the used tools for evaluation and the methodology. The obtained results are presented in section 5. The related work is presented in section 6 and the paper concludes with section 7.

2. DSE Basic Concepts

In multi objective optimization there are usually multiple conflicting objectives that need to be simultaneously optimized. Because of this an order between individuals (solutions to the problem) can not be established easily. An individual can perform better on one objective compared to another individual but worse on the other.

[1] http://www.intel.com/ - September 2009 22nm silicon technology SRAM test chips. In released processors: 1,17 bilion Intel® Core™ i7 Processor at a 32nm technology.

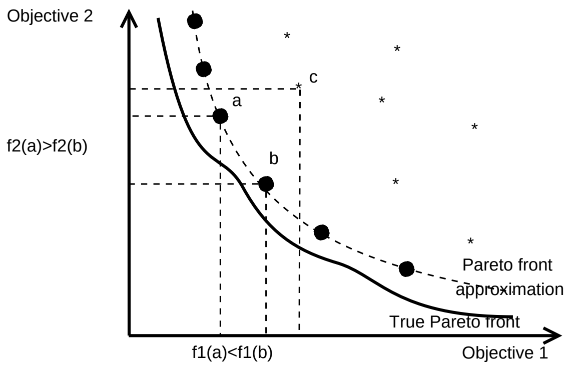

Fig.1 True Pareto front and the Pareto approximation

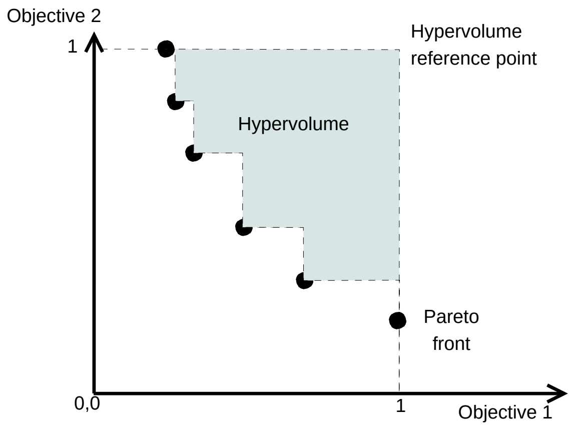

Fig.2 Hypervolume computation for a problem with two objectives

In Figure 1 some notions from the Pareto efficiency are depicted. It presents the true Pareto front and the approximated Pareto front for a multi objective problem with two objectives. The representation is in the objective space. The true Pareto front (sometimes called Pareto front) contains the optimal individuals. These individuals are called nondominated, because no relation can be established between them. The Pareto front approximation contains the individuals found by the DSE algorithm. They can belong to the true Pareto front (if the individuals are optimal) or not. These individuals are also nondominated with respect to the current individuals. For example point *a* is better on objective 1 but worse than *b* on objective 2 (this is a minimization problem). So no relation can be established between these two points and they are called nondominated. On the other hand point *c* is dominated on both objectives by point *a* and *b* and it is called dominated.

The metrics used in this article are hypervolume and coverage. They were chosen because the true Pareto front, which is unknown in real life problems, is not needed to compute them.

The hypervolume measures the space covered by the approximated Pareto front and the axes in a maximization problem. FADSE solves only minimization problems so the axes and their origin can not be used. A point has to be chosen (see Figure 2) called hypervolume reference point. This point is situated at the maximum coordinates, found from the entire Pareto front approximation, on all the objectives. The hypervolume value is equal with the hyperarea covered by the Pareto front approximation from the hypercube obtained using the hypervolume reference point and the axes. The value is normalized to one (see Figure 2).

The second metric is called coverage of two sets and it can be used to compare two sets of individuals. The coverage of two sets metric returns how many individuals from one set dominate individuals from another set.

Since the NSGA-II algorithm is used extensively in this article a short presentation is given. NSGA-II is a genetic algorithm developed by Deb et. al.[2]. The algorithm generates a first initial population and evaluates it. It then performs crossover and mutation on this initial population and obtains the offspring population. The algorithm evaluates the offspring population and performs a union between the two populations (initial population, called parent population and the offspring). Using the nondomination concept it sorts the union. The individuals who are nondominated with respect to one another are further sorted by their crowding measurement (density computation) and a fitness value is assigned. The best individuals are selected according to their fitness. These selected individuals will form the new parent population.

The other algorithms used are SPEA2 and SMPSO. SPEA2 is similar with NSGA-II. SPEA2 developed by Zitzler et. al.[3] (as presented in Calborean et. al.[4]) uses an external population (called archive) to keep nondominated individuals. It also assigns fitness in a different manner: the strength of each individual is computed (this is equal with the number of individuals the current individual dominates). Then the raw fitness is computed which is the sum of strengths of the individuals that dominate current individual. To this raw fitness, a density information is added (the inverse distance to the nearest k^{th} neighbor) to obtain the fitness of the individual. The algorithm then selects the best individuals (from both the archive and the offspring population) and stores them in the archive. The archive is then used as a parent population.

SMPSO is a particle swarm optimization method. The particle swarm optimization (PSO) algorithms (as presented by us in our previous article Calborean et. al.[4]) are inspired by the flight of birds while trying to find food. In these algorithms the population is called swarm. The individuals are called particles. The particles are "flown" through space following the best performing particle at that moment. The position of a particle is given by the current values of its parameters, belonging to an orthogonal representation in particle's space. As the particle tries to get closer to the current best particle its parameters are modified. The change takes into account both the current global best (the particle with the highest fitness) and the particle's personal best (the position in space where the current particle performed best), who act as attractors. Based on this, the particle gets a new position and it needs to be evaluated. After all the particles are evaluated, the new global best particle is selected, the personal bests are updated and the process is restarted. In multi-objective PSO algorithms there can be multiple global best particles.

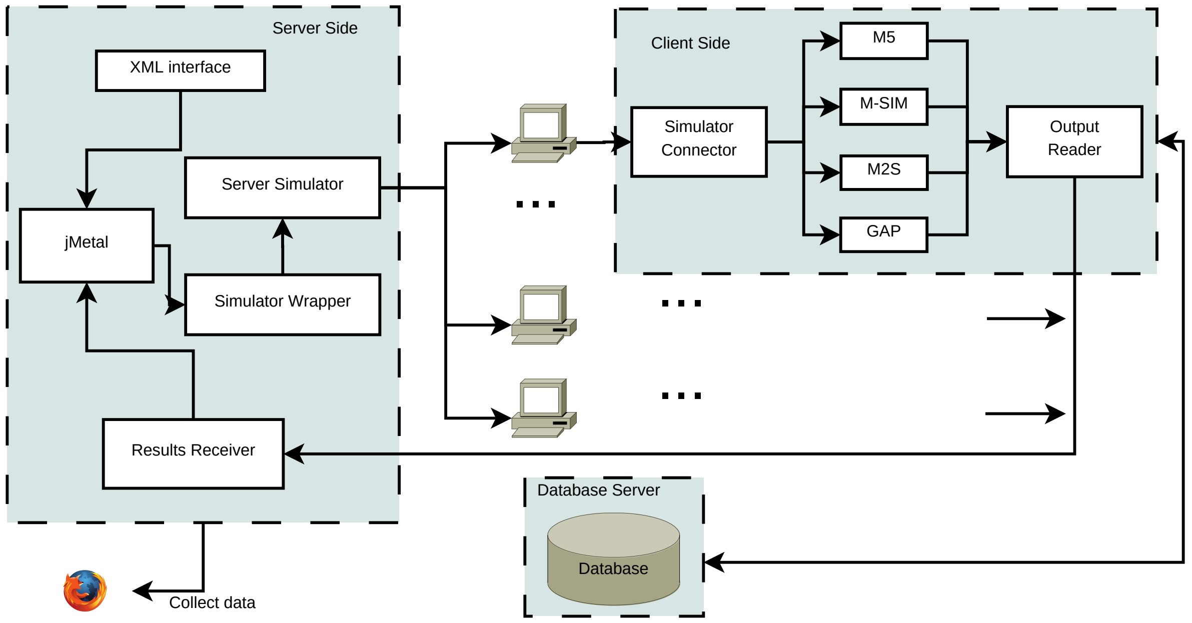

Fig. 3 FADSE application structure

3. The Developed Framework

During the past year we have developed a framework called FADSE. It has already been presented in some of our previous articles: Calborean and Vintan [5] [6] and Jahr et. al.[4]. An overview of its features is also presented here.

FADSE is a tool which performs automatic design space exploration using state of the art evolutionary algorithms from the jMetal library developed by Durillo et. al.[7]. The focus of FADSE is on computer architectures and because of its modular and extensible structure almost any existing computer simulator can be integrated with it. We have written connectors for simulators like: M5[2], Multi2Sim Ubal et. al.[8], M-SIM2, M-SIM3[3], GAP (and its code optimization tool - GAPtimize) Uhrig et. al.[9].

Evaluating a single design point can take a lot of time (hours) and usually the existent simulators are not able to run the simulations in parallel. Because of this FADSE was developed to allow parallel evaluation of the design points. Evolutionary algorithms usually generate an entire population and evaluate the individuals one by one. But the results obtained are not used until all the individuals are evaluated. Because of this, the whole process can be done in parallel. The entire population can be sent for evaluation asynchronously and in parallel, thus, if resources are available, decreasing the overall simulation time. To assure correctness a join method has to be called before using the results. We had to modify the algorithms included in jMetal.

To support parallel evaluation FADSE is built as a client-server application (see Figure 3). When the server is started it configures the algorithm from jMetal using the data provided by the user in the input XML (detailed below). The algorithm then begins generating possible configurations (called individuals in this paper). The individuals are sent to the ServerSimulator module which is responsible of distributing the work on the clients. The server discovers a client which is able to receive the current individual and sends it for evaluation. It then takes the next individual and tries to send it to the next free client. This process continues until all the individuals are sent. If there are no more free clients the server waits for a client to become free.

When a client receives an individual it searches in the database to see if it has been already simulated in the past. This database increases the speed of the search through reuse. Over the course of 100 generations using the NSGA-II Deb et. al. [2] algorithm with a population of 100 (0.9 crossover probability and 0.16 mutation probability) we have seen a reusability of individuals of about 60%. If the individual has not been evaluated before (it is not found in the database) the simulator is started. At the end of the simulation the results are saved in the database and then sent back to the server. The server collects all the results and pushes them back to the algorithm which can now create the next generation.

FADSE can run with genetic algorithms like NSGA-II, SPEA2 introduced by Zitzler et. al.[3], particle swarm optimization algorithms (OMOPSO introduced by Sierra and Coello [10], SMPSO by Nebro et. al.[11]) and many other.

FADSE runs successfully on various systems: from commodity networks (LANs) to supercomputers (University of Augsburg 32 processing cores) and HPC systems (ULBS grid).

The design exploration process usually runs for days/weeks. Running for long periods of time makes it vulnerable to different types of faults in the system: loss

[2] http://www.m5sim.org/

[3] http://www.cs.binghamton.edu/~msim/

of power, network problems and bugs in the simulators. To prevent failures different mechanisms have been adopted:

A watchdog timer has been implemented on the clients so if a client encounters problems and is unable to receive messages for a long period of time it is automatically restarted.

Checkpointing has also been implemented. If the server has a problem (power loss for example) the execution can be continued from where it has remained.

The server can also decide if a client has exceeded its simulation time and decide to resend the same individual to another client. After a number of retries the individual is considered infeasible.

Clients can be stopped or reintroduced in the system at any time, the server being able to cope with these situations.

The framework can be easily configured using an XML file. In this XML file the simulator connector has to be configured. Common parameters are: path to the simulator, path to the output file, maximum time of a simulation. The database server, the DSE algorithm and its properties are configured next. The supported database management system (DBMS) is MySQL, but any other relational DBMS that supports SQL language can be easily used. A list of benchmarks can be specified. FADSE is able to evaluate each benchmark on a different client and compute the average on the server when all the simulations are done.

In the XML file the parameters that need to be varied are configured. Parameters can be of type: arithmetic progression (with different ratios), geometric progression (with a ratio of 2) and list of strings.

Rules can be imposed between parameters. These are used to constrain the design space and/or avoid impossible configurations. An example of such rules is shown below:

```
<rule name="l2 cache minimum size">
    <greater-equal>
        <parameter name="  dl2_nsets*
                           dl2_bsize*
                           dl2_assoc"/>
        <parameter name="262144"/>
    </greater-equal>
</rule>
```

This rule limits the minimum size of the level 2 cache to 256KB. This can be use to constrain the design space. The rules can be used to avoid impossible configurations such as the one in the example below:

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

This second rule imposes that the level 2 cache size has to be bigger than the level 1 data cache added with the level 1 instruction cache. Arithmetic operations can be used between the parameters.

The status of the exploration can be queried by sending commands to a configurable port on witch FADSE listens for external commands. This feature allows the user to monitor the DSE process and in the future to influence it in real time.

After the results are obtained different metrics can be used to evaluate the design exploration. FADSE includes some of the most used metrics that do not require the true Pareto front to be known: hypervolume, coverage, two set difference hypervolume and the so called “7 Point” average distance (described by Coello et. al.[12]).

4. Methodology and tools

In our research we have used FADSE with different simulators. Results were obtained with M-SIM3 simulator, M-SIM2 simulator, the GAP simulator and its code optimization tool GAPtimize. In this article the focus will be on the results obtained with GAP.

GAP (Grid Array Processor) simulator is developed at the University of Augsburg and a more detailed presentation can be found in the articles by Uhrig et. al.[13] [14] Jahr et. al.[15] and Caborean et. al.[4]. GAP was evaluated on 10 selected benchmarks from the MiBench suite presented by Guthaus et. al. [16]. The benchmarks were compiled with GCC with optimizations turned on (-O3). The design space has a size of over $1.1 * 10^6$ individuals.

Two objectives were used in the optimization: CPI and complexity Jahr et. al.[15].

The algorithms ran were: NSGA-II, SPEA2 and SMSPO. For the genetic algorithms (NSGA-II and SPEA2) we have used the single point crossover operator and the bit flip mutation operator. The population and the archive size were set to 100. For the crossover operator a probability of 0.9 was used and for mutation a probability of 1/(number of parameters) as recommended in Deb et. al.[2]. For the particle swarm optimization algorithm (SMPSO) we have used the following settings (recommended in Nebro et. al.[11]):

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

We started all the algorithms from the same random initial population. In this way we have a common base in our comparisons. If the algorithms were started from different initial populations then multiple runs would have been required to provide a fair comparison. This is not feasible because of long evaluation times.

5. Evaluation

In this section the results obtained with the NSGA-II algorithm on the GAP simulator are presented. Then a comparison between NSGA-II, SPEA2 and SMPSO performed by Calborean et. al.[4] is reviewed.

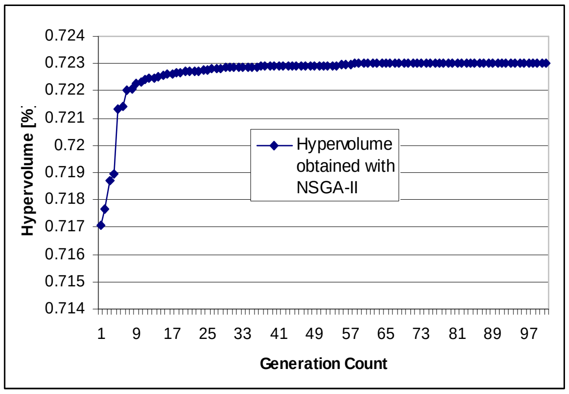

Fig.4 Hypervolume obtained with NSGA-II during 100 generations on the GAP simulator

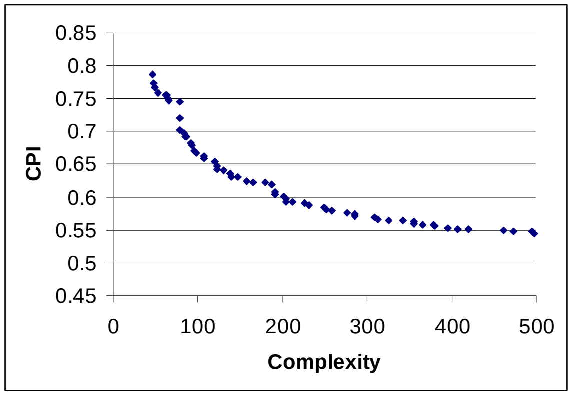

Fig.7 Pareto front approximation obtained by NSGA-II on GAP simulator after 50 generations.

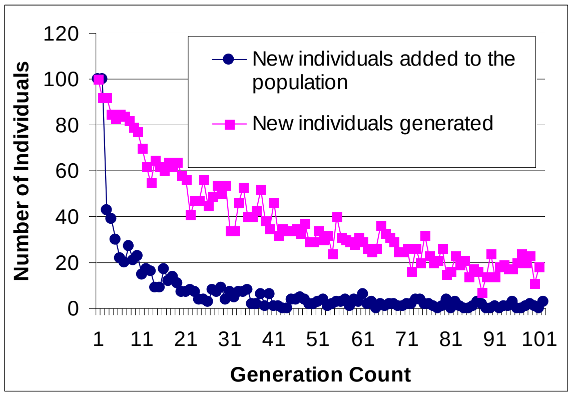

Fig.5 Evolution of the number of generated/accepted individuals for NSGA-II on the GAP simulator

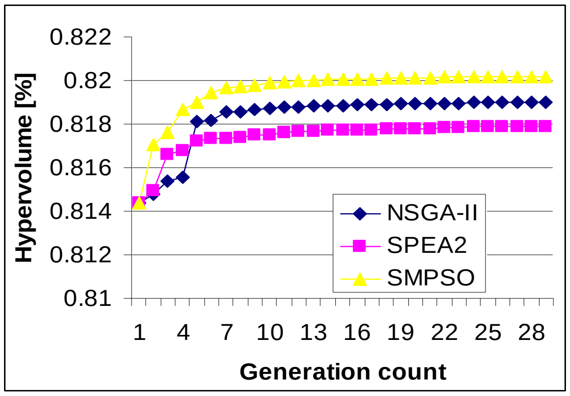

Fig.8 Hypervolume comparison between NSGA-II, SPEA2 and SMPSO

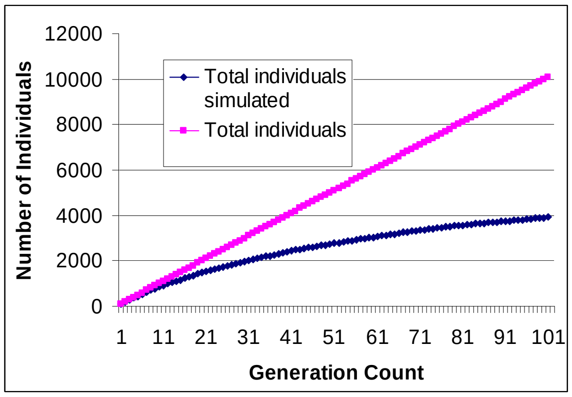

Fig.6 Number of simulated individuals with database reuse compared to number of individuals that would have been simulated if no database was present.

In the work performed by Silvano et. al. [17] the NSGA-II genetic algorithm is pointed out as the best performing algorithm from the selected ones. Therefore we chose NSGA-II in our evaluation too and compared it with other two algorithms (SPEA2 and SMPSO) not included in the work performed by Silvano et. al.

For the first evaluations we used only the NSGA-II algorithm and the results are presented below.

In Figure 4 the hypervolume value obtained over the generations is depicted. It can be observed that the algorithm converges fast for the first 10 generations and then after generation 25 the progress made is small. It can be concluded that the algorithm is unable to find better solutions as the generation count increases and it has converged. This information can be used to stop the algorithm: when no changes on the hypervolume value are observed for a number of generations the algorithm can be stopped. This will be done automatically in future versions of FADSE.

Figure 6 shows how many new individuals (not generated before) are generated in each offspring population and how many of them are added to the parent population after the selection process. It can be observed that for the first generations there are many new individuals and also many of them are better than the individuals from the parent population so they get selected and reach the next parent population. Figure 6 is correlated with Figure 4: as the algorithm moves forward it is not able to produce new individuals so even fewer of them are better than the ones already present in the parent population thus the hypervolume value stops increasing.

Because FADSE has been integrated with a DBMS it can reuse already simulated individuals. As the generation count increases and the number of new individuals decreases only a small percentage of the individuals from the offspring population has to be simulated, the rest can be obtained from the database. The distance in Figure 6 between the total individuals and total individuals simulated shows a graphical representation of the above statement. To reach generation 100 with a population of 100 individuals, only 3912 from a total of 10000 were sent for

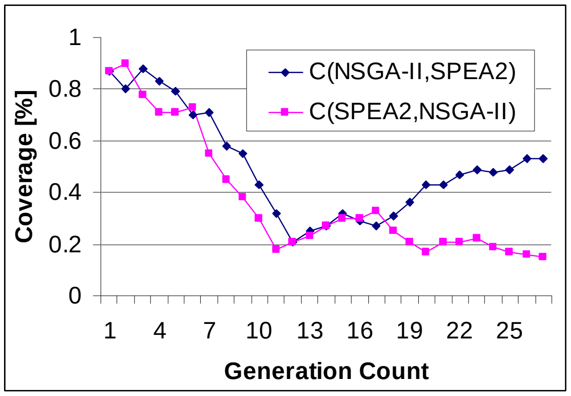

Fig.9 Coverage comparison between NSGA-II and SMSPO

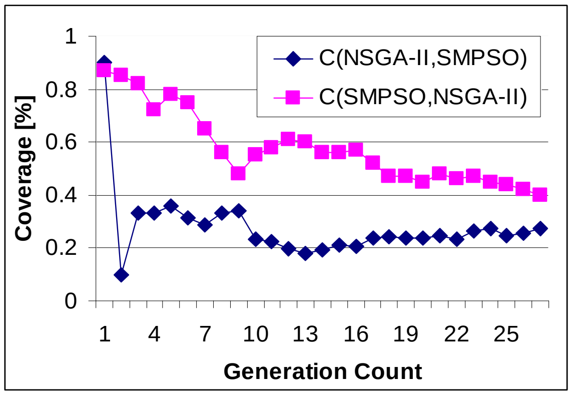

Fig.10 Coverage comparison between NSGA-II and SMSPO

simulation. The introduction of the database proves to be of great importance and reduces the DSE process time considerably. These conclusions were presented by us in Jahr et. al. [15]

In Figure 7 the obtained Pareto front approximation is depicted. We have analyzed the results and compared them with previous manually obtained results. We can conclude that FADSE is able to find better results than the ones obtained by a human designer. The results obtained by FADSE had half of the complexity at the same CPI Jahr et. al.[15].

In Calborean et. al.[4] a comparison between NSGA-II, SPEA2 and SMPSO was performed. In this article we present those results, too. In Figure 8 a comparison is performed between the hypervolume values obtained by the three algorithms on the GAP simulator. It can be easily observed that SMSPO provides the best results and also has the fastest convergence.

In Figure 9 and 10 the same algorithms are compared using the coverage metric. First NSGA-II is compared with SPEA2 (Figure 9). NSGA-II clearly performs better and so it is then compared with SMPSO (see Figure 10). Again SMPSO provides better results confirming the conclusion drawn from the hypervolume comparison. In Calborean et. al.[4] we have compared the algorithms from the perspective of number of simulations required. The conclusion was that both NSGA-II and SPEA2 tend to stop producing new individuals after a certain number of generations. SMPSO does not preserve the behavior of NSGA-II and SPEA2, and it tends to generate new individuals at each generation, thus reducing the advantage of the reuse. When the quality of results is compared after the same amount of simulations is done, SMPSO still performs better.

6. Related work

There are other existing tools which perform automatic design space exploration using evolutionary algorithms. An overview is presented here Saxena and Karsai [18]. We will describe the most notable ones.

M3Explorer by Silvano et. al. [17] is a DSE framework, part of the Multicube FP7 project [19], very similar with our tool. FADSE has been developed trying to maintain compatibility at the interface level with the M3Explorer. M3Explorer also contains many DSE algorithms and allows the user to write a specific connector. It lacks the distributed evaluation present in FADSE.

Archexplorer.org Desmet et. al. [20] is a website where anyone can upload a module that simulates a component of the system (only memory at this moment) and it will be automatically integrated. The user defined module enters in an “online competition” with similar modules and it is evaluated over time. Archexplorer uses statistical exploration with genetic operators. The DSE algorithm cannot be changed by the user.

Another tool is Magellan Kang and Kumar [21] which focuses on multi-core architectures but it is bound to one simulator that can not be easily changed.

7. Conclusions

In this article we have performed an overview of our previous work and also an analysis of the database integration influence on the number of simulated individuals. We have observed a great reduction in number of simulations and in time required for the DSE process. From around 4 hours per generation on a supercomputer with 32 cores at the beginning of the search process to around 20 minutes per generation as the search approached generation 100.

We have shown that FADSE can provide very good results, better than the ones obtained through manual design space exploration.

FADSE has proven to be very flexible, allowing the user to choose from a multitude of DSE algorithms and to use this tool with his/hers own simulator. From our experience writing a connector between FADSE and another simulator does not require more than a couple of hours of work.

In the future we will integrate FADSE with UniMap Radu and Vintan [22] [23].

We will continue to develop FADSE to help the algorithms find faster better results through the use of domain knowledge.

Acknowledgements

Horia Calborean was supported by POSDRU financing contract POSDRU 7706.

The authors would like to thank Ralf Jahr and Professor Theo Ungerer for providing access to the Augsburg 32 processing cores computer.

References

[1] L. Vintan, “Directii de cercetare in domeniul sistemelor multicore / Main Challenges in Multicore Architecture Research,” *Revista Romana de Informatica si Automatica*, vol. 19, no. 3, 2009.

[2] 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.

[3] E. Zitzler, M. Laumanns, L. Thiele, and others, “SPEA2: Improving the strength Pareto evolutionary algorithm,” in *Eurogen*, 2001, p. 95–100.

[4] H. Calborean, R. Jahr, T. Ungerer, and L. Vintan, “Optimizing a Superscalar System using Multi-objective Design Space Exploration,” presented at the The 18th International Conference On Control Systems And Computer Science, Bucharest Romania, 2011.

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

[6] H. Calborean and L. Vintan, “Toward an efficient automatic design space exploration frame for multicore optimization,” in *ACACES 2010 poster Abstracts*, Terassa, Spain, 2010, pp. 135-138.

[7] J. J. Durillo, A. J. Nebro, F. Luna, B. Dorronsoro, and E. Alba, *jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics*. E.T.S.I. Informatica, Campus de Teatinos: Departamento de Lenguajes y Ciencias de la Computacion, University of Malaga, 2006.

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

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

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

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

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

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

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

[15] R. Jahr, H. Calborean, T. Ungerer, and L. Vintan, “Automatic Multi-Objective Optimization of Parameters for Hardware and Code Optimizations,” *submited to HPCS11 Istanbul, Turkey*, Jul. 2011.

[16] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, “MiBench: A free, commercially representative embedded benchmark suite,” in *wwc*, 2001, p. 3–14.

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

[18] T. Saxena and G. Karsai, “A Meta-Framework for Design Space Exploration,” presented at the Engineering of Computer-Based Systems, Las Vegas, USA, 2011.

[19] M. O. D. S. E. OF and M. P. SOC, “FP7-216693-MULTICUBE Project.”

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

[21] S. Kang and R. Kumar, “Magellan: a search and machine learning-based framework for fast multi-core design space exploration and optimization,” in *Proceedings of the conference on Design, automation and test in Europe*, Munich, Germany, 2008, pp. 1432-1437.

[22] C. Radu and L. Vințan, “Optimized Simulated Annealing for Network-on-Chip Application Mapping,” presented at the International Conference on Control Systems and Computer Science, Bucharest, Romania, 2011 in press.

[23] C. Radu and L. Vințan, “Optimizing Application Mapping Algorithms for NoCs through a Unified Framework,” in *Roedunet International Conference (RoEduNet), 2010 9th*, Sibiu, Romania, 2010, pp. 259 - 264.