Optimizing a Superscalar System using Multi-objective Design Space Exploration

Horia Calborean [*] **Ralf Jahr** [**] **Theo Ungerer** [**]
Lucian Vintan [*]

[*] *Lucian Blaga University of Sibiu, Computer Science & Engineering Department, E. Cioran Str., No. 4, Sibiu - 550025, Romania (e-mail: horia.calborean, lucian.vintan@ulbsibiu.ro).*
[**] *University of Augsburg, Institute of Computer Science, Universitaetsstr. 6a, 86159 Augsburg, Germany (e-mail: jahr, ungerer@informatik.uni-augsburg.de)*

Abstract:
One way to cope with a huge design space formed by several parameters is using methods for Automatic Design Space Exploration (ADSE). Recently we developed a Framework for Automatic Design Space Explorations focused on micro-architectural optimizations.
In this article we evaluate the influence of three different evolutionary algorithms on the performance of design space explorations. More precisely, we selected two genetic algorithms, NSGA-II and SPEA2, as well as the bio-inspired SMPSO algorithm, running a particle swarm optimization. With these algorithms we run FADSE to optimize both the parameters of the Grid ALU Processor (GAP) microarchitecture and of the GAPtimize post-link code optimizer. An analysis of the simulation results showed a very good performance of the SMPSO algorithm during the design space exploration process on the GAP simulator. SPEA2 provided slightly better results than NSGA-II when they were used on the design space exploration of both GAP and GAPtimize.

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

1. INTRODUCTION

In last years a growth in the complexity of computer systems could be observed. The number of architectural parameters has risen, leading to a huge number of possible configurations. As an example, if there are 50 parameters each being able to have one of eight possible values, it involves 8^{50} number of possible configurations. Simulating all these configurations to find the best possible solutions would take extremely long time for representative benchmarks. The time-to-market requirements will not allow this.

The current design methodology implies a designer who, based on his/her experience, manually chooses parameters for the architecture. As an alternative solution for this hard job automated search tools have been built to help the user finding good configurations.

Since optimizing a system is usually a multi-objective problem (most of the times with conflicting objectives) the task of finding a good solution becomes even harder.

In order to do this automatically, we have developed a framework, called Framework for Automatic Design Space Explorations (FADSE) introduced by Calborean and Vintan (2010b), Calborean and Vintan (2010a) and Jahr et al. (2011). FADSE is able to perform automatic design space explorations using state-of-the-art evolutionary and bio-inspired multi-objective algorithms.

In this article we compare three well known heuristic algorithms: two genetic algorithms (NSGA-II and SPEA2) and one particle swarm optimization (SMPSO) algorithm, optimizing the Grid ALU Processor (GAP). We also compare the two genetic algorithms in a design space exploration of both GAP and the code optimization tool called GAPtimize.

These algorithms have been compared in other works, too (this will be presented in the related work section), but in totally different contexts (synthetic optimization problems). The obtained results are strongly dependent on the specific problem. As far as we know we are the first trying to determine which of these algorithms finds the best solutions in the least amount of time on processor architectures in conjunctions with a code-optimizer.

The article is structured as follows: Section 2 provides an overview of the related work. In section 3, some basic concepts about design space exploration are presented as well as the metrics used in our experiments. Section 4 offers a short presentation of the DSE algorithms. The tools (FADSE, GAP and GAPtimize) are presented in section 5 along with the objectives used and the parameters for the DSE algorithms. The results of our evaluation are presented in section 6 and finally section 7 concludes the paper and presents some further work.

2. RELATED WORK

There is not too much work on comparing different algorithms on computer architecture simulators. Such a comparison is made by Silvano et al. (2010) with the M3Explorer. The authors compare some algorithms including NSGA-II and a particle swarm optimization algorithm by Palermo et al. (2008). NSGA-II is clearly the best performing algorithm from the selected ones and outperforms the particle swarm optimization algorithm. The design space of their exploration consists of only 9126 possible configurations which allowed them to perform an exhaustive evaluation. In Kang and Kumar (2008) a set of single objective algorithms (genetic algorithms, ant colony optimization, hill climbing, random search, etc.) is compared against an exhaustive simulation. The search space is greatly reduced by fixing many parameters to allow them an exhaustive search. The conclusion is that the genetic algorithm is able to reach a solution 0.1% worse than the best possible solution but 23000 times faster. Still, the work is not extended to multi-objective optimization.

The authors of NASA Jia et al. (2010) perform a DSE using single objective genetic algorithms for each objective. Multiple algorithms are used at the same time and a comparison between sets of these algorithms is made.

There are also many articles comparing algorithms not on real world problems but on synthetic ones. For example, in Nebro et al. (2009) SMPSO is compared with NSGA-II and SPEA2. The conclusion which the authors reach is that SMPSO clearly outperforms NSGA-II and SPEA2. In Zitzler et al. (2001) a comparison between SPEA2 and NSGA-II is performed but no clear winner is found.

The metrics used in all of these articles require the knowledge of the optimal solutions. This is not feasible in real world problems; in our target problem this is also true.

From the perspective of DSE on optimizing computer architectures there are some other existing tools: Arch-explorer by Desmet et al. (2010) is a web-tool which allows users to (up)load implementations of their computer architecture components (e.g. caches) to be optimized. The modules have to respect a special interface so they can be integrated in the DSE framework. The type of components that can be integrated is limited to the available interfaces. The algorithm used for DSE is statistical exploration with genetic operators. The DSE algorithm cannot be changed and also no comparison between different types of algorithms is performed by the authors.

3. DSE BASIC CONCEPTS

3.1 Multi-objective optimization problem

In many of the real world optimization problems there are multiple conflicting objectives. An order between individuals (solutions to the problem) cannot be established easily. One individual can be better than another individual on one objective but worse on another one. Therefore the notions from the concept of Pareto efficiency are used. In Fig. 1 points a and b are called nondominated because no order can be established between them: a is on objective 1 better than b but worse on objective 2 (minimization problem). Point c is worse than a and b on both objectives, so it is dominated. The set of nondominated solutions is called Pareto set approximation. In the objective space they form the Pareto front approximation. The optimal solutions, which are unknown in real life problems, form the Pareto set (called true Pareto front in objective space).

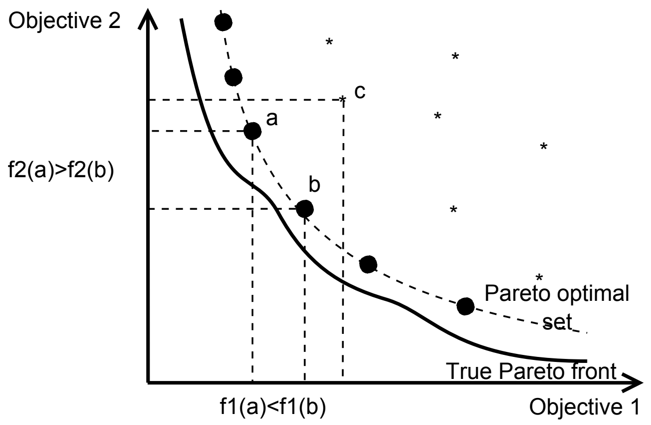

Fig. 1. Pareto front approximation and true Pareto front. Points a and b belong to the Pareto front approximation because they are nondominated (none of them is better than the other on both objectives). Point c is dominated by both point a and b.

3.2 Metrics used

To be able to perform an unbiased comparison of the three algorithms metrics are needed. Many of the metrics found in literature require the true Pareto front to be known. Since the search space is enormous an exhaustive exploration (simulating all the possible configurations) can not be performed to discover the optimal solutions, hence they can be used only for simple synthetic test problems with known true Pareto front (like the DTLZ family of problems Deb et al. (2002b)). Due to our problem's complexity, we had to select some metrics that do not require the true Pareto front to be known, so they can be used to observe the DSE's progress. The selected metrics are widely used in literature.

Coverage by Coello et al. (2002) can be used to compare two multi-objective algorithms, or two runs of the same algorithm (with the same or different parameters). It returns the fraction of individuals produced by one algorithm that dominates individuals produced by the other algorithm.

Hypervolume by Coello et al. (2002), also known as hyperarea, computes the volume enclosed by the current Pareto front approximation and the axes, for a maximization problem. If it is computed at each generation, the evolution of this volume shows if the algorithm makes progress. Also if multiple algorithms are compared it can show the quality of the results of one algorithm over the other. For a minimization problem a point has to be established, called hypervolume reference point, which will replace the origin in the hypervolume computation. The hypervolume reference point (HV) is set to the coordinates of the maximum values of the objectives (see Fig. 2). If multiple algorithms are compared the maximum values have to be searched from all the different runs so that the

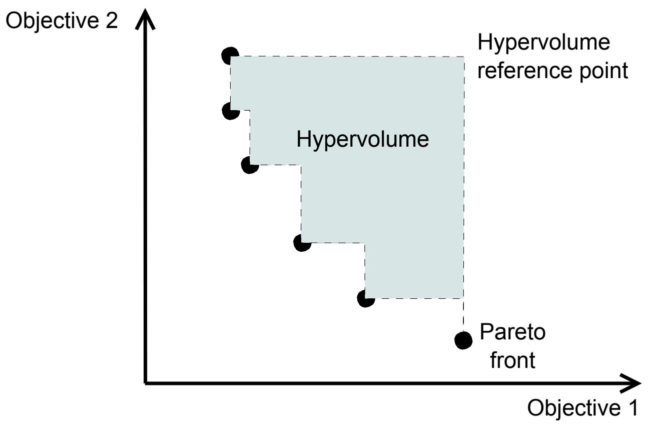

Fig. 2. Calculus of the hypervolume for a minimization problem with two objectives

area enclosed between the axes and the HV remains fixed. The values returned by the hypervolume computation are normalized to the interval [0,1] using this constant area.

4. COMPARED ALGORITHMS

4.1 NSGA-II

NSGA-II is a genetic algorithm developed by Deb et al. (2002a) . The algorithm generates a first initial population and evaluates it. On this initial population it performs crossover and mutation to obtain the offspring population. The offspring population is evaluated. The two populations are composed into a single one and sorted according to the domination relationship. The individuals which are nondominated with respect to one another are further sorted by their crowding measurement (density computation). Then the best individuals are selected according to their fitness assigned after the sorting procedure. These selected individuals form the next population.

4.2 SPEA2

SPEA2 introduced by Zitzler et al. (2001) is another genetic algorithm. The only differences between NSGA-II and SPEA2 are: (a) SPEA2 uses an external population (archive) to keep nondominated individuals and (b) it assigns fitness in a different manner. First the strength of each individual is computed (which is equal with the number of individuals the current individual dominates). Then the raw fitness is computed. The raw fitness is the sum of strengths of the individuals that dominate current individual. To this raw fitness, a density information is added which is the inverse distance to the nearest k^{th} neighbor to obtain the fitness of the individual. The algorithm then selects the best individuals (from both the archive and the offspring population) and stores them in the archive. The archive is then used as a parent population.

4.3 SMPSO

SMPSO by Nebro et al. (2009) is a particle swarm optimization method (bio-inspired) developed from OMOPSO. The particle swarm optimization (PSO) algorithms 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, which are "flown" through space following the best performing particle at that moment. The position of a particle is given by the current values of its parameters, belonging to an orthogonal representation particle's space. As every particle tries to get closer to the current best particle its parameters are changed. The change takes into account both the current global best and the particle's personal best. Based on this change the particle gets a new position and it needs to be evaluated again. After all the particles are evaluated, the new global best particle is selected, the personal bests are updated and the process is restarted. In multi-objective PSO algorithms there can be multiple global best particles.

5. METHODOLOGY AND TOOLS

In this section we present FADSE, the design space exploration tool we have developed. The GAP simulator and the code optimization tool GAPtimize are also introduced in more detail. At the end of the section the DSE process parameters and their values are enumerated.

5.1 FADSE

FADSE (Framework for Automatic Design Space Exploration) allows distributed design space exploration of computer systems using multi-objective state-of-the-art design space algorithms. It was already presented in Calborean and Vintan (2010b) and in Calborean and Vintan (2010a). Since the structure of the application has changed and many improvements have been made the current state is presented below.

FADSE was designed so that almost any existing computer system simulator can be connected to it by writing a specific connector. We have implemented connectors to computer architecture simulators like: GAP and GAPtimize (used in this article), M-SIM3, M-SIM2 [1], M5 [2] and Multi2Sim v2 by Ubal et al. (2007).

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

We have changed some of the algorithms implemented in jMetal by Durillo et al. (2006) to work in a distributed manner. Most of the evolutionary algorithms generate an offspring population and only after all the individuals are evaluated they are used. Taking advantage of this behavior the evaluation process can be distributed easily. The offspring individuals are evaluated in parallel by FADSE. If multiple benchmarks need to be run to evaluate a single configuration then these separate evaluations are also done in parallel. FADSE then collects all the results for a single individual and computes the average. As an example, for an algorithm with an offspring population of 100 and where each individual has to be evaluated on 10 benchmarks we reach 1000 simulations that can be run in parallel.

[1] `http://www.cs.binghamton.edu/~msim/`
[2] `http://www.m5sim.org/`

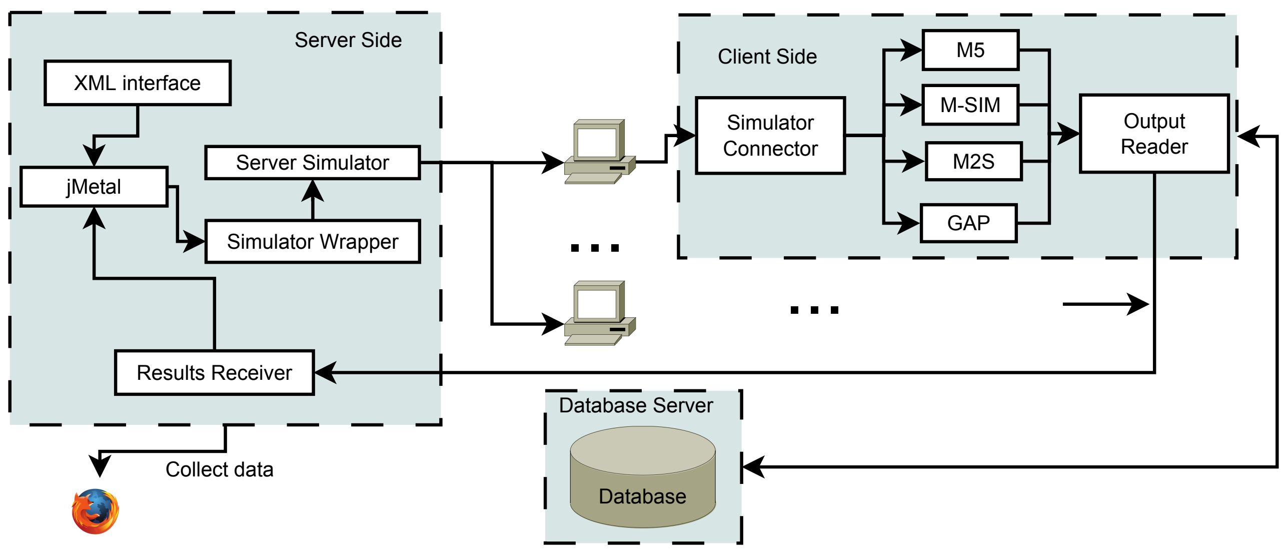

Fig. 3. FADSE structure of the application

FADSE has been tested successfully on a LAN of Windows machines, on a Windows based computer with 32 processor cores and on a Linux based HPC system (30 Intel Xeon quad-cores). The parallelism can be exploited down to the core level of the host machine (several simulations can be run on the same computer). FADSE maintains a queue of free clients (clients which are currently not simulating) and at the time it needs to perform a simulation it pulls the first client and assigns it the task. When a client completes a simulation is added at the end of the queue.

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

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

A database (currently MySQL) can be used to store the results of the simulations. The evolutionary algorithms tend to produce some of the individuals again during an exploration process. When the same individual is generated the results can be extracted from the database, instead of redoing the simulations, leading to a much shorter exploration time. In our experiments, on 100 generations with a population of 100 individuals per generation in a design space of around 1.1 million configurations with the NSGA-II algorithm, a reuse factor of 67% was observed.

In parallel with the server, a monitor thread is started. The monitor can be accessed through an open port and the status of the server can be queried. This can help the user to track the progress of FADSE, the status of the simulations, etc.

A XML interface is used to configure FADSE, the design space and the specific constrains. First the user has to specify the simulator connector he/she wants to load and the set of parameters required by the connector. Then a list of benchmarks is specified, the database connection is configured and the list of parameters that need to be varied (these parameters can be of type integer, arithmetical progression, geometrical progression, list of strings, etc.) is specified.

A list of application-specific rules can be described in the XML, too. These rules are used to constrain the design space. Different rules can be designed: from simple conditions (e.g. L2 cache must be larger than L1 cache) to more complex ones. A user can, for example, construct a composed rule like the one below:

```
IF l2cache > 2*l1d_cache AND num_cores <= 4 THEN
l3cache > 4096.
```

More details about the XML interface can be found in Calborean and Vintan (2010b), Calborean and Vintan (2010a) and Jahr et al. (2011).

The design space exploration process starts by loading the input XML file (see Fig. 3). FADSE configures itself and selects the desired search algorithm. The DSE algorithm generates individuals and sends them for simulation to the SimulatorWrapper. The SimulatorWrapper validates every individual against all constraints and if it is valid sends it to the ServerSimulator. The ServerSimulator checks if there is a free clients and tries to send the individual to one. When a client receives an individual it loads the desired connector and, if the individual is not in the database, starts the simulation with the parameters specified by the individual. After the results are obtained (either after the simulation or reused from the database) the individual is sent back to the server and the process is restarted.

5.2 Grid ALU Processor

The Grid ALU Processor (GAP) introduced by Uhrig et al. (2010) is a novel processor architecture designed to speed up the execution of sequential instructions streams. Its basis is a superscalar processor with in-order execution. The main change is replacing the execution units by an array of functional units (FUs). Instructions are mapped onto this structure dynamically and at runtime by an additional stage of the frontend, the configuration unit. Therefore, it is not necessary to re-compile a program to be able to execute it on the GAP; a common instruction set as for a superscalar processor can be used.

To buffer configurations, i.e. instruction sequences placed onto the array, a structure called *configuration layers* with large similarities to a trace cache has been introduced. With it the latencies for issuing instructions can be reduced effectively and hence the total system performance increases.

The GAP is very scalable because one can choose for the size of the array of FUs any size between 4x4 and 31x32. In the design space exploration we want to find good sizes for the array (height, width) in connection with the number of configuration layers (1, 2, 4, ..., 64) as well as the size and organization of the instruction cache.

Two objectives were defined for the GAP so far. The first is the total system performance gained from a cycle accurate simulator. The number of clock cycles per instruction (CPI) is used as measure. As second objective we introduced a model (see Jahr et al. (2011)) to make the complexity of the hardware of two different configurations of the GAP comparable. This is referred to as complexity in the following. To it contribute mainly the instruction cache, whose size is calculated with CACTI, the number of FUs and the number of configuration layers. Both objectives are to be minimized.

5.3 GAPtimize, Post-Link Optimizer for GAP

With GAP's ability to execute legacy code, the best point to apply code-optimizations, which make use of platform-specific features, is a post-link optimizer. GAPtimize has been developed for this. It supports at the moment e.g. branch predication, an optimization called static speculation Jahr et al. (2010) and inline expansion of functions.

In this paper, inlining is used as a prototype for a code optimization. Our heuristic has four parameters which form the parameter space to be explored by FADSE.

As objective function the number of clock cycles per reference instruction is used (CPRI). This is very similar to CPI described above. The number of clock cycles for the program with code optimizations is not divided by the actually number of instructions executed (as it is with CPI), because this can vary. Instead it is divided by the number of instructions executed by the program if no code optimization is used. The optimization goal is the same as for CPI; smaller is better.

5.4 Parameters of the DSE algorithms

For all the DSE algorithms we used the same mutation and crossover (if required) operators, i.e. bit flip mutation and single point crossover. The selection operator for the genetic algorithms was binary tournament selection as described in Deb et al. (2002a). The mutation probability was set for all the algorithms to 1/(number of decision variables). The crossover probability was set to 0.9. The distribution index was 20 for both mutation and crossover. These values are commonly used for the evolutionary algorithms. The population size (swarm size for particle swarm optimization) was set to 100 for all the evaluations and the archive size for SPEA2 and SMPSO was also set to 100, as recommended in Deb et al. (2002a).

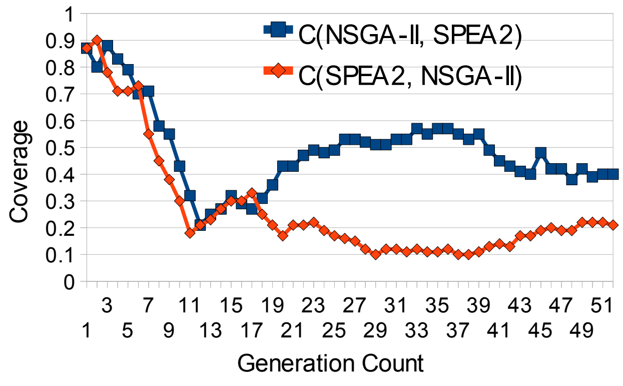

Fig. 4. Coverage comparison between NSGA-II and SPEA2.

For the velocity computation in SMPSO the following values were used (as recommended in Nebro et al. (2009)):

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

We have generated a random initial population and all the algorithms start from this initial population. This provides a fair comparison of the algorithms. Without this, multiple runs would be needed and an average result should be presented to provide unbiased results. Multiple runs were infeasible due to time constraints.

6. EVALUATION

In this section we present the evaluation results for all the algorithms on the GAP processor and respectively on GAP with GAPtimize.

6.1 FADSE with GAP

The DSE process on GAP was done with 6 microarchitectural parameters on 10 benchmarks selected from the MiBench suite. The benchmarks were compiled with GCC with optimizations turned on (`-O3`). The design space has a size of over $1.1 * 10^6$ individuals. The DSE was stopped after 50 generations.

We first compute the coverage and compare the two genetic algorithms (see Fig. 4). It can be observed that NSGA-II performs better than SPEA2. The evolution is close for the first 17 or 18 generations but then NSGA-II clearly performs better.

A coverage comparison between NSGA-II and SMPSO was made (see Fig. 5) and SMPSO clearly performs better than NSGA-II for all the generations (since NSGA-II is better than SPEA2 the coverage between SMPSO and SPEA2 is not presented). It can be observed in Fig. 5

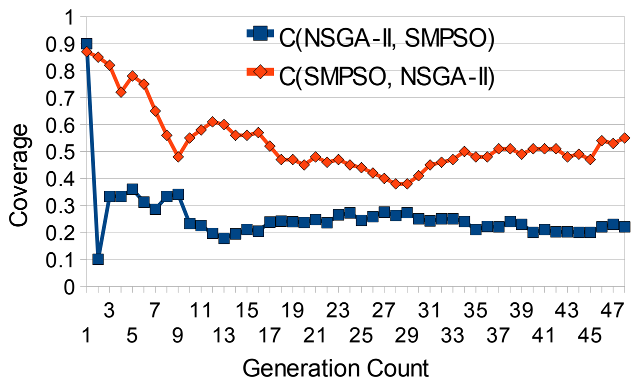

Fig. 5. Coverage comparison between NSGA-II and SMPSO.

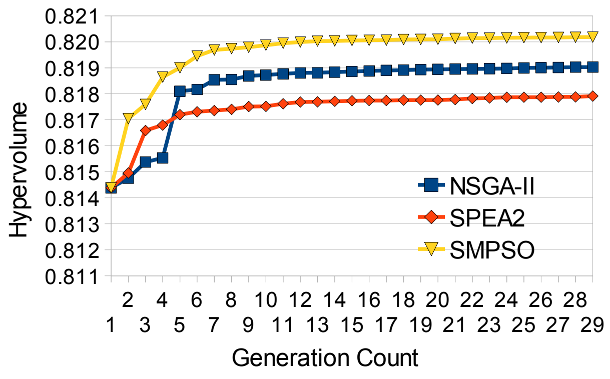

Fig. 6. Hypervolume comparison between NSGA-II, SPEA2 and SMPSO.

that the coverage difference between SMPSO and NSGA-II is extremely high at the second generation. This might happen because SMPSO converges faster and obtains good results and NSGA-II requires some time to reach the same quality of configurations.

Next we compare the algorithms using the hypervolume metric. The hypervolume not only shows us the quality of the results but, if its trend over the generations is analyzed, it shows the speed of the convergence of the algorithms. From Fig. 6 it can be observed that SMPSO is still the best performing algorithm from the selected three: it finds the best solutions (highest value of the hypervolume) and also SMPSO converges the fastest to a better Pareto front approximation (hypervolume values rise faster than for the other algorithms).

Since in design space exploration of computer architectures we are dealing with long evaluation times, it is necessary to compare the number of distinct evaluations required by each algorithm to reach a certain generation. In Fig. 7 it can be observed that SPEA 2 requires less evaluations for 50 generations. SPEA2 tends to retain many duplicates in its archive during the environmental selection process (see Zitzler et al. (2001)) because of the density computation method. In SPEA2 identical individuals will have the same fitness assigned, while in NSGA-II these individuals can have a different fitness assigned, because of the crowding assignment Deb et al. (2002a). In SPEA2 there is a big chance that both individuals get selected and passed to

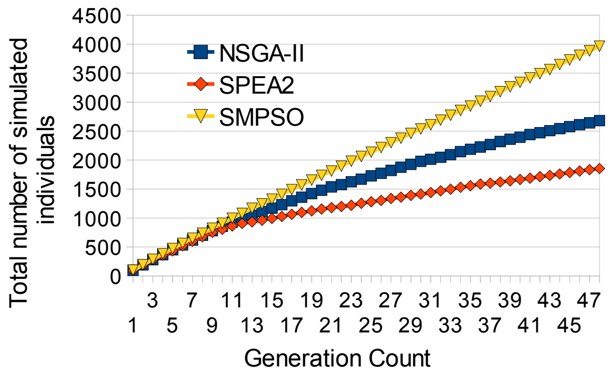

Fig. 7. This figure shows how many individuals were simulated to reach a certain generation.

the next generation while in NSGA-II one of them might be discarded. At the selection process all the individuals have the same chance of becoming parents but since in SPEAs archive there are more duplicates than in NSGA-II they have a greater chance of being selected and they produce the same individuals. The advantage is that these individuals do not have to be simulated again due to the integrated database. The disadvantage is that SPEA2 does not have such a good spread of solutions (even if at the end there are 100 individuals many of them are duplicates). SMPSO has a different approach since it does not create new individuals/particles and it only "flies" them through space. When a particle is moved all its parameters are changed. This behavior of PSO algorithms prevents generating the same positions for a certain individual so often, but also increases the number of required simulations.

With this in mind, it is interesting to compare the hypervolume against the number of simulated individuals. In Fig. 8 we can see that the conclusion remains the same. SMPSO has the best performance from this point of view, too. At the same number of simulated individuals the hypervolume value of the Pareto front approximation, discovered by the particle swarm optimization algorithm, is superior to those obtained by the other two genetic algorithms.

The final results (after 50 generations) are shown in Fig. 9. The figure presents a portion of the Pareto front approximation. We have selected only this area because on the rest of the obtained Pareto front approximation there are no visible differences between the algorithms. It can be observed that SMPSO indeed finds better configurations but only on a small area, between a complexity of 100-180, while the complexities of all the solutions found by the algorithms range from 30 to 2000 (not shown in the figure).

6.2 FADSE with GAP and GAPtimize

In this section we present the results obtained after FADSE was run on GAP and GAPtimize with NSGA-II and SPEA2.

We used the same 10 benchmarks as in the previous section, but this time they were compiled without function inlining since GAPtimize is used for function inlining. The

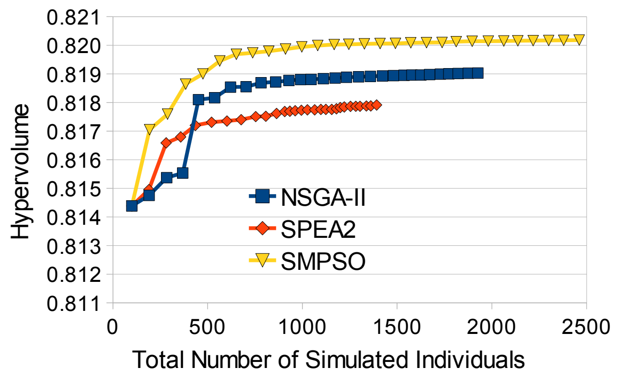

Fig. 8. Hypervolume comparison of the three selected algorithms against the total number of evaluated designs

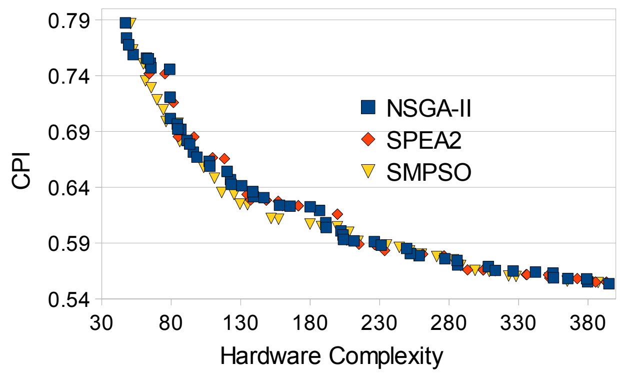

Fig. 9. Section of the Pareto front approximations obtained by each algorithm after 50 generations.

cache and the number of configuration layers for GAP were fixed and only the size of the array was varied. For GAPtimize four parameters were changed. The design space with these 6 parameters has a size of around $1.6 * 10^{13}$.

We have done a hypervolume comparison between NSGA-II and SPEA2 on this problem, too. From the hypervolume we conclude that SPEA2 performs better than NSGA-II. Also SPEA2 continues to perform fewer simulations than NSGA-II. We present in Fig. 10 the hypervolume evolution against the total number of simulations.

The coverage computation (see Fig. 11) also shows that SPEA2 performs much better than NSGA-II when the design space exploration process is performed on both GAP and GAPtimize.

NSGA-II was stopped after 30 generations. To reach generation 30 NSGA-II had to simulate around 1800 individuals. We continued the run of SPEA2 until the same number of simulations were performed (10 benchmarks * 1800 individuals). SPEA2 reaches generation 68 before evaluating 1800 individuals. A comparison of the Pareto front approximations obtained after 1800 evaluations is shown in Fig. 12. The results obtained by SPEA2 are slightly better than the results obtained by NSGA-II thus confirming the concussions drawn from the two metrics (hypervolume and coverage).

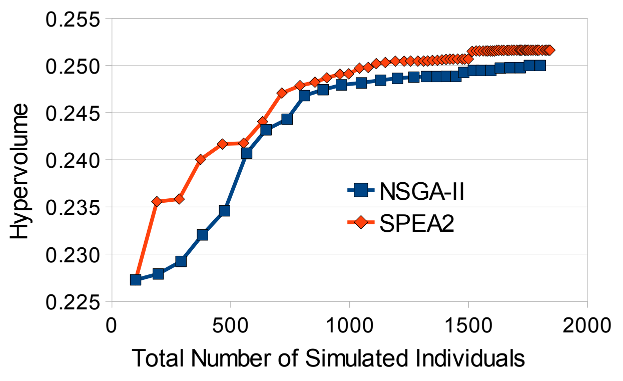

Fig. 10. Hypervolume comparison between NSGA-II and SPEA2.

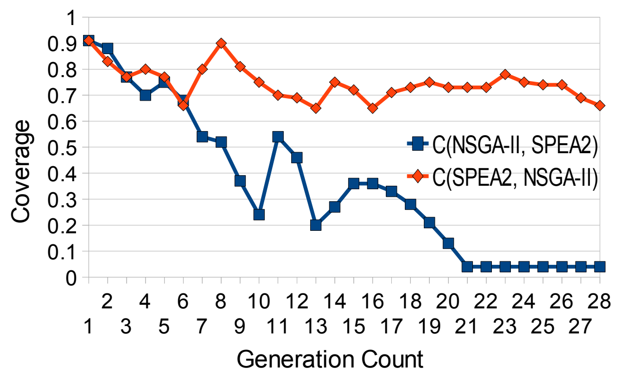

Fig. 11. Coverage comparison between DSE runs with NSGA-II and SPEA2.

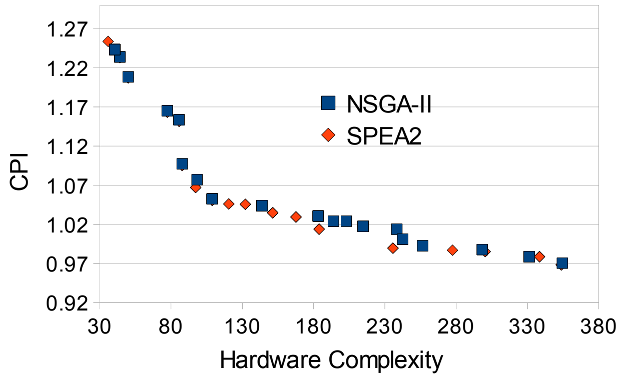

Fig. 12. The final Pareto front approximations obtained by NSGA-II and SPEA2 after 1800 simulations

7. CONCLUSIONS AND FURTHER WORK

We have already proven in Jahr et al. (2011) that evolutionary algorithms are able to find good results, better than the ones found by a human expert (NSGA-II was used). If we look at the Pareto front approximations obtained by all the evaluated algorithms (see Fig. 9 and Fig. 12) the difference between the best found individuals in terms of performance is not greater than 1% (and usually around 0.1-0.01%) at the same complexity. All the algorithms are able to find good results, so the factor that needs to decide which one proves to be best is its convergence speed. The results obtained for GAP show the best convergence for the PSO algorithm, thus confirming the results obtained on synthetic problems by Nebro et al. (2009) in their comparison of SMPSO with NSGA-II and SPEA2. A clear recommendation between NSGA-II and SPEA2 can not be made. NSGA-II performed better than SPEA2 on GAP but worse on GAP with GAPtimize. But the spread, observed after looking at the obtained Pareto front approximations, of the solutions found by SPEA2 is worse than the one obtained by NSGA-II.

We can also conclude that the coverage metric can be misleading by showing a big difference between the algorithms while in fact the results are only slightly better. So even if the quality of solutions seems much better, in reality there is not a great difference. In a future article metrics that use e-dominace as presented by Coello et al. (2002) could be used, which should avoid these pitfalls. Hypervolume does provide a fair image and it helps discovering which of the algorithms converges faster and when does the algorithm stop discovering better solutions.

Due to time constrains this article has presented a comparison of only three algorithms. In a further article this work will be extended to more algorithms. A comparison between different particle swarm optimizers will be made. Algorithms that were especially developed for problems with long evaluation times will be also analyzed in the future.

In our evaluations, all algorithms were started from the same random population. If some known good individuals would have been included not at random but because of domain knowledge it would potentially help the algorithm find good solutions faster.

Fuzzy control logic will be used to input domain knowledge information into the algorithms. The user is currently able to describe different relations between parameters from within FADSE. Membership functions can be associated to a parameter. These relations together with the rules form a domain ontology and could potentially help the design space exploration algorithms perform better.

ACKNOWLEDGEMENTS

Lucian Vintan was supported by CNCSIS no. 485/2008 research grant offered by the Romanian National Council for Academic Research. Horia Calborean was supported by POSDRU financing contract POSDRU 7706.

REFERENCES

Calborean, H. and Vintan, L. (2010a). Toward an efficient automatic design space exploration frame for multicore optimization. In *ACACES 2010 poster Abstracts*, 135–138. Terassa, Spain.

Calborean, H. and Vintan, L. (2010b). An automatic design space exploration framework for multicore architecture optimizations. In *Roedunet International Conference (RoEduNet), 2010 9th*, 202 –207. Sibiu, Romania.

Coello, C.A.C., Veldhuizen, D.A.V., and Lamont, G.B. (2002). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 1 edition.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002a). A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2), 182–197.

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2002b). Scalable multi-objective optimization test problems. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, 825–830.

Desmet, V., Girbal, S., Ramirez, A., Temam, O., and Vega, A. (2010). Archexplorer for automatic design space exploration. *Micro, IEEE*, 30(5), 5 –15.

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

Jahr, R., Shehan, B., Uhrig, S., and Ungerer, T. (2010). Static speculation as post-link optimization for the grid alu processor. In *Proceedings of the 4th Workshop on Highly Parallel Processing on a Chip (HPPC 2010)*.

Jahr, R., Ungerer, T., Calborean, H., and Vintan., L. (2011). Automatic multi-objective optimization of parameters for hardware and code optimizations. *accepted for publication HPCS11 Istanbul, Turkey.*

Jia, Z.J., Pimentel, A., Thompson, M., Bautista, T., and Nunez, A. (2010). Nasa: A generic infrastructure for system-level MP-SoC design space exploration. In *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on*, 41 –50.

Kang, S. and Kumar, R. (2008). Magellan: a search and machine learning-based framework for fast multicore design space exploration and optimization. In *Proceedings of the conference on Design, automation and test in Europe*, 1432–1437. ACM, Munich, Germany.

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

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

Silvano, C., Fornaciari, W., Palermo, G., Zaccaria, V., Castro, F., Martinez, M., Bocchio, S., Zafalon, R., Avasare, P., Vanmeerbeeck, G., et al. (2010). MULTICUBE: Multi-Objective Design Space Exploration of Multi-Core Architectures. In *Proceedings of the 2010 IEEE Annual Symposium on VLSI*, 488–493. IEEE Computer Society.

Ubal, R., Sahuquillo, J., Petit, S., and López, P. (2007). Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors.

Uhrig, S., Shehan, B., Jahr, R., and Ungerer, T. (2010). The two-dimensional superscalar GAP processor architecture. *International Journal on Advances in Systems and Measurements*, 3(1 and 2), 71 – 81.

Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.