www.ietdl.org

Published in IET Computers & Digital Techniques
Received on 9th August 2011
Revised on 19th January 2012
doi: 10.1049/iet-cdt.2011.0116

ISSN 1751-8601

Multi-objective optimisations for a superscalar architecture with selective value prediction

A. Gellert   H. Calborean   L. Vintan   A. Florea

Computer Engineering Department, 'Lucian Blaga' University of Sibiu, Emil Cioran Street, No. 4, Sibiu 550025, Romania
E-mail: arpad.gellert@ulbsibiu.ro

Abstract: This work extends an earlier manual design space exploration (DSE) of the authors' developed selective load value prediction-based superscalar architecture to the $L2$ unified cache. After that the authors perform an automatic DSE using a special developed software tool by varying several architectural parameters. The goal is to find optimal configurations in terms of cycles per instruction and energy consumption. By varying 19 architectural parameters, as the authors proposed, the design space is over 2.5 millions of billions configurations which obviously means that only a heuristic search can be considered. Therefore the authors propose different methods of automatic DSE based on their developed framework for automatic design space exploration which allow them to evaluate only 2500 configurations of the above mentioned huge design space! The experimental results show that their automatic DSE provides significantly better configurations than the previous manual DSE approach, considering the proposed multi-objective approach.

1 Introduction

This paper presents an automatic design space exploration (DSE) of an architecture containing a selective load value prediction (SLVP) scheme suitable for energy-aware superscalar processors. A load value predictor is a hardware architectural enhancement which speculates over the results of load instructions to speed-up the execution of the subsequent instructions. Lipasti first proposed the load value prediction (LVP) concept and, in particular, he developed a non-selective LVP in 1996 but we have not used his LVP structure owing to its huge complexity. Our proposed architectural enhancement (SLVP) differs from a classic value predictor owing to an improved selection scheme that allows activating the predictor only when a miss occurs in the first level of cache. We use a simple direct-mapped table which requires less additional hardware enabling reduced energy consumption than traditional approaches (Lipasti's first proposal).

We first used our SLVP in [1] where we have analysed the efficiency of selectively anticipating the results of long-latency instructions within superscalar and simultaneous multithreaded (SMT) architectures. In particular we have focused on multiply, division and critical loads (with miss in $L1$ data cache). We integrated into the M-SIM simulator [2] a dynamic instruction reuse scheme for the Mul/Div instructions and an LVP for the critical load instructions. Our improved superscalar architecture achieved an average instruction per cycle (IPC) speed-up of 3.5% on the integer SPEC 2000 benchmarks, of 23.6% on the floating-point benchmarks, and an improvement in energy-delay product of 6.2 and 34.5%, respectively. Our evaluations have also shown higher IPC and lower relative energy consumption (energy-delay product) on all the evaluated SMT configurations (1, 2, 3 and 6 threads).

Since our previous results show that most of the IPC speed-up was generated by the load value predictor we further focalised on this speculative technique. In [3] we performed a manual DSE regarding the size of the $L1$ data cache in order to find the optimal configuration, which keeps high performance at low energy consumption. We have shown that the performance lost by reducing the $L1$ cache capacity can be covered by our SLVP technique. The experimental results, performed on the SPEC 2000 benchmarks, have expressed that reducing the $L1$ data cache space by quartering its size and using SLVP produces an improvement of the IPC and energy consumption in both the superscalar and SMT architectures against the corresponding baseline architectures.

In this work we extend the manual DSE of a SLVP-based superscalar architecture to the $L2$ unified cache. Our goal is to find optimal configurations in terms of cycles per instruction (CPI) and energy consumption. After these manual design space explorations performed by varying only two parameters we intend to increase the number of the varied parameters to 19. By varying 19 architectural parameters, as we proposed, the design space is over 2.5 millions of billions configurations which obviously means that only a heuristic search can be considered. Therefore we propose different methods of automatic DSE based on our developed framework for automatic design space exploration (FADSE) tool which allow us to evaluate around 2500 configurations of the above mentioned huge design space, while still finding good solutions! We implemented a domain ontology consisting of some micro-architectural restrictions and expert knowledge expressed through fuzzy rules, in order to accelerate the design space exploration. By performing a multi-objective automatic DSE of the same architecture (using our developed FADSE tool [4]), but varying 19 architectural parameters, the obtained configurations are significantly better than the manually obtained ones.

The paper is organised as follows. Section 2 reviews the state-of-the art of value prediction techniques and some basic concepts about design space exploration. Section 3 introduces the target architecture and presents the simulation methodology. Also, a short presentation of the used metrics is performed. Sections 4 and 5 describe the manual and automatic design space exploration, respectively, together with experimental results obtained on the Alpha AXP 21264 architecture. Finally, Section 6 summarises the relevant contributions of this work and presents some further work directions.

2 Related work

Lipasti *et al.* [5] originally introduced LVP as a new data-speculative micro-architectural technique exploiting the concept of value locality and the dynamic correlation between load instruction address and its actual value. An important difference between our value prediction approach and Lipasti's is that we selectively predict only the load instructions that generate a miss in *L*1 cache. Thus, we attenuate the mispredictions cost and reduce the hardware cost of the speculative micro-architecture. Moreover, since less hardware is required, there is also less power consumption.

Other value predictors like the stride-, context- and perceptron based, have been proposed in our earlier work [6] to register centric value prediction. The SLVP was already used in our previous papers [1, 3] and other load value predictors have been proposed in [7, 8]. Different architectural support techniques for value prediction or energy efficient approaches that speculate on the results of load instructions in order to speed-up the execution are presented in [9, 10]. In [9] the authors selectively re-execute only the speculatively retired instructions that depended on the mispredicted value, so called ForwardSlice, performing a speed-up execution with an additional hardware budget (ReSlice buffer). Somewhat similar, in [10] the author proposes a new load latency tolerant design that is both energy efficient, and applicable to both in-order and out-of-order cores, based on slice re-execution as an alternative use of multi-threading support, efficient schemes for register and memory state management, using a chained store buffer for efficient store-to-load forwarding and using pruning mechanisms to reduce re-execution overheads. The main idea of load latency tolerance is to virtually scale the critical execution structures (issue queue, physical register file). Load latency tolerance designs remove from these window resources all dependent instructions of loads with miss in caches in order to allow younger instructions to enter the pipeline and execute. When a miss returns, the instructions which depend on it are re-injected into the pipeline – re-acquiring issue queue entries and physical registers – and re-executed.

Further, in this section, we present some of the best known DSE tools. M3Explorer [11] is a DSE framework that includes many DSE algorithms. M3Explorer can use response surface models to accelerate DSE. Another DSE tool is in the form of a website: archexplorer.org [12]. The users can upload their component on the website where it is integrated into a computer system simulator. The design is compared against other designs introduced by other users. The users do not have any control on the algorithm being used. NASA [13] is also a similar tool. It allows the user to easily integrate his/her DSE algorithm and offers the possibility to connect to any simulator (features offered also by FADSE). Magellan [14] is a DSE tool which is bound to a certain simulator (SMTSIM). Magellan can perform only single objective DSEs.

In [15] we used our developed FADSE tool to explore the vast design space of the grid alu processor (GAP) and its post-link optimiser called GAPtimize, both developed at Augsburg University. It has shown that FADSE is able to thoroughly explore the design space for both GAP and GAPtimize and it can find an approximation of the Pareto [4, 16] frontier consisting of near-optimal individuals in moderate time. For the GAP, FADSE can find, owing to the approximation of the complexity, efficient configurations.

To our knowledge FADSE is the single DSE tool that allows the user to introduce domain knowledge through fuzzy rules, written in a human-readable form, in order to accelerate DSE. Mariani *et al.* [17] use neural networks to accelerate the DSE process. The authors predict through neural networks if an individual is worth simulating or not, but still the knowledge of an architect is not used.

3 Simulation methodology

All the experimental results presented further were obtained using SPEC 2000 benchmarks on 500 million dynamic instructions, skipping the first 300 million instructions. We evaluated six floating-point benchmarks (*applu*, *equake*, *galgel*, *lucas*, *mesa*, *mgrid*) and six integer benchmarks: computation intensive (*bzip*, *gcc*, *gzip*) and memory intensive (*mcf*, *twolf*, *vpr*). Our measurements are generated using an 80 nm CMOS technology and 1.2 GHz frequency.

The target architecture is a superscalar Alpha AXP 21264 processor augmented with a direct mapped SLVP of 1024 entries, access latency of one cycle and prediction latency of three cycles [3]. It has a register file of [32 int/32 fp]*8, a reorder buffer (ROB) of 128 entries and a load/store queue (LSQ) of 48 entries. First-level caches are 64 KB, 2-way associative, with a 1-cycle latency. The second-level unified cache is 4 MB, 8-way associative and 6-cycle latency. The main memory has a latency of 100 cycles.

For the performance metrics we chose CPI (and not IPC) because we want to minimise all the objectives for the clarity of the Pareto graphs. For the relative CPI reduction we used the following formula

$$\mathrm{CPI}_{\mathrm{reduction}} = \frac{\mathrm{CPI}_{\mathrm{base}} - \mathrm{CPI}_{\mathrm{improved}}}{\mathrm{CPI}_{\mathrm{base}}} 100\,[\%] \tag{1}$$

where $\mathrm{CPI}_{\mathrm{base}}$ and $\mathrm{CPI}_{\mathrm{improved}}$ are CPI with the baseline and improved architectures, respectively. A positive value of $\mathrm{CPI}_{\mathrm{reduction}}$ means a performance improvement related to the baseline architecture.

The detailed power modelling methodology, used in the simulator, is presented in [18]. The dynamic power consumption in CMOS microprocessors is defined as

$$P = CV_{\mathrm{dd}}^2 af \tag{2}$$

where C is the capacitance, generated using *Cacti* [19], V_{dd} is the supply voltage and f is the clock frequency. V_{dd} and f depend on the assumed process technology. The activity factor a indicates how often clock ticks lead to switching activity on average. The power consumption of the modelled units highly depends on the internal capacitances of the circuits. From the capacitance point of view, there are three categories of architectural structures: array structures, content-associate memories and complex logic blocks. The first two categories are used to model the caches, branch predictors, the reorder buffer, the register renaming table and the register file, whereas the last category is used to model functional units.

For the power consumption evaluation we used the ‘aggressive non-ideal conditional clocking model’ [20] which scales linearly the power of active units with their usage and assumes 10% power dissipation in the case of unused units. The instantaneous average power consumption (P_{Mean}) for a certain benchmark is computed with the following relation

$$P_{\text{Mean}} = \frac{\int_0^{\text{T}} P(t)\,\mathrm{d}t}{T} \tag{3}$$

where T is the total simulation time in cycles and P is given in relation (2). The energy consumption is given by

$$E = P_{\text{Mean}} T \tag{4}$$

where P_{Mean} is computed with relation (3). The average energy (weighted mean) is given by the following formula in [$W\cdot$ cycles]

$$E_{\text{Mean}} = \frac{\sum_{i=1}^{N} E_i T_i}{\sum_{i=1}^{N} T_i} \tag{5}$$

where N is the number of benchmarks, E_i is the total or per unit energy computed for benchmark i and T_i is the total number of cycles executed within benchmark i. The energy reduction percentage is given by

$$E_{\text{reduction}} = \frac{E_{\text{base}} - E_{\text{improved}}}{E_{\text{base}}} 100\,[\%] \tag{6}$$

where, E_{base} and E_{improved} are the energy consumptions of the baseline and our improved architectures, respectively. Thus, a positive value of $E_{\text{reduction}}$ means an improvement of the relative energy consumption.

4 Manual DSE of the unified *L*2 cache

A method to increase the cache performance is to reduce the penalty in case of misusing multilevel caches. Our simulated architecture uses two level exclusive caches. This allows smaller *L*2 data caches involving less power consumption. The evictions are performed based on the least recently used (LRU) algorithm for both cache levels.

The first goal of our research consists in performing a DSE regarding the sizes of the *L*1 data cache and the unified (instruction and data) *L*2 cache in superscalar architectures augmented with SLVP structures. Thus, we will double, halve, quarter and eighth the *L*2 cache and we will halve, quarter and eighth the *L*1 data cache, considering as reference the architecture presented in Section 3. We note with mUL2_$_n$DL1 a configuration using m^*4 MB 8-way associative unified *L*2 cache ($m = 2$, 1, 1/2, 1/4, 1/8) and n^*64 KB 2-way associative *L*1 data cache ($n = 1, 1/2, 1/4, 1/8$).

Fig. 1 presents the relative CPI and energy reduction – computed based on formulas (1) and (2), respectively, – of different configurations with SLVP of 1024 entries reported to the baseline configuration without SLVP. It can be observed that the SLVP helps in maintaining a better CPI and energy consumption when the cache sizes are reduced. The CPI reduction with the help of the SLVP is positive up to using halve of UL2 and eighth of DL1. Starting with quartering UL2, the CPI reduction is negative; therefore no performance improvement is achieved.

The energy reduction is lower in the case of reducing the *L*2 cache to 1/8 than in the case of quartering it. The energy consumption has a static and a dynamic component. 1/8UL2 cache implies a higher miss rate than 1/4UL2 and therefore higher dynamic power consumption, owing to the higher number of off-chip accesses. Thus, even if the static power consumption of 1/8UL2 is lower than that of 1/4UL2 the energy consumption is higher owing to higher dynamic power consumption. Therefore using only the quarter of the *L*2 cache (2 MB) and the eighth of the *L*1 data cache (8 KB) is optimal from the energy consumption viewpoint.

As a preliminary conclusion, after the manual DSE the best configuration regarding CPI is 2UL2_DL1 whereas the best configuration in terms of energy consumption is 1/4UL2_1/8DL1. There are also some optimal configurations from both CPI and energy viewpoints: 1/2UL2_1/2DL1 and 1/2UL2_1/4DL1. These results obtained through manual DSE encourage us to explore a larger design space by automatic DSE because the best and the optimal configurations are different and there are also other parameters which can be varied.

5 Automatic DSE

In the previous section we varied only the cache sizes through our manual DSE. Besides the caches there are several

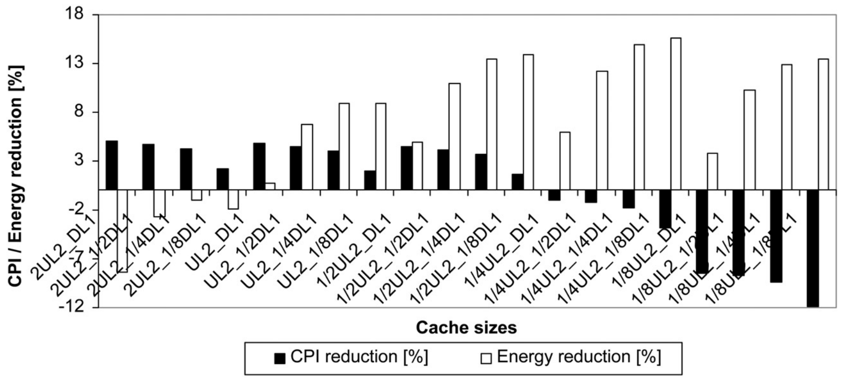

Fig. 1 *Relative CPI and energy reduction reported to UL2_DL1 without SLVP as baseline*

parameters that can highly influence our two objectives: CPI and energy consumption. We selected 19 important architectural parameters to be varied during our automatic DSE, with the lower and upper limits given in Table 1. By varying these 19 architectural parameters the design space grows over $2.5*10^{15}$ (2.5 millions of billions) configurations which obviously means that only a heuristic search can be considered. Therefore we propose different methods of automatic DSE based on our developed FADSE tool that also contains an NSGA-II genetic algorithm implementation.

To perform DSE we have developed a tool called FADSE. It includes many state-of-the art evolutionary algorithms through the included jMetal [21] library. FADSE can be connected to almost any existing simulator. The parameters are described through an extensible XML interface. FADSE allows parallel evaluation (included algorithms had to be modified to allow this).

FADSE is a client–server application. The number of clients can be dynamically changed. Clients can be stopped or started while the DSE process runs. Since performing DSE can take a lot of time (weeks), reliability of the DSE tool is a major concern. FADSE is able to cope with failing clients, failing networks or even power loss of the entire system. It is able to recover from these situations by detecting the problems and resubmitting the simulations to other clients. In case of power loss, it can restart the DSE process by making use of the integrated checkpointing mechanism. It contains a database which allows reusing already simulated individuals. This leads to a reduction of the time required to perform an exploration process. FADSE includes many metrics that the user can choose to evaluate the DSE process or to compare different algorithms. Some of the implemented metrics are: hypervolume, coverage, two set difference hypervolume [22] etc.

We have chosen for our automatic DSE the NSGA-II genetic algorithm. NSGA-II is a multi-objective genetic algorithm developed by Deb *et al.* [23]. NSGA-II has been chosen as it provided very good results in our previous experiments. In [16] we showed that NSGA-II, SMPSO [24] and SPEA2 [25] obtain results of similar quality but SPEA2 does not have such a good spread of solutions when optimising the GAP system. In [4] we compared NSGA-II, SPEA2, SMPSO and OMOPSO [22] on the UniMap simulator [4]. We showed that NSGA-II and SPEA2 obtain better results, in terms of quality, than the other two algorithms. It starts from a random population called the parent population. From this parent population an offspring population is generated by means of mutation and crossover (see below). The two populations (offspring and parent) are merged into a single one and the best individuals (architectural configurations) are selected according to their fitness value. The fitness value is computed considering the domination relationship and a density function. These individuals will form the new parent population and the process is repeated.

Table 1 Parameter limits

Parameter		Lower limit	Upper limit
DL1/IL1 cache	sets	2	32768
	block size (bytes)	8	256
	associativity	1	8
UL2 cache	sets	256	2097152
	block size (bytes)	64	256
	associativity	2	16
SLVP (entries)		16	8192
decode/issue/commit width		2	32
ROB/LSQ/IQ size (entries)		32	1024
number of physical register sets (int/fp)		2/2	8/8
int/fp ALU		2	8
int/fp MUL/DIV		1	8

NSGA-II is not a distributed algorithm by default. To accelerate the DSE process we have changed the algorithm and now the individuals are evaluated in parallel. This is possible because the values of the objectives of an individual are required only after all the individuals are evaluated. So an entire population can be evaluated in parallel and a single synchronisation point has to be established at the end of a generation. We configured the NSGA-II algorithm as follows:

Stop condition: we will observe the hypervolume progress. If there is no progress for at least X generations we consider that the algorithm has converged. To measure the progress we will use the following formula

$$\text{Progress} = \sum_{i=1}^{X} (H_k - H_{k-i}) \qquad (7)$$

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

Population size: 100 – as recommended in [23].

Mutation: bit flip mutation with a mutation probability of 1/nparam [23] (where nparam is the number of varied parameters). In our situation the mutation probability is set to 0.05 (19 parameters).

Crossover: single point crossover, probability of crossover set to 0.9 (as specified in [23]).

Selection operator: binary tournament selection (described in [23]).

Used metrics:

- Hypervolume: in a maximisation problem the hypervolume is the volume enclosed between the Pareto front approximation and the axes. In a minimisation problem a point has to be selected (called hypervolume reference point). The hypervolume reference point is selected at the coordinates provided by maximum values of the objectives.
- Other metrics: number of generated individuals, comparisons between the obtained Pareto fronts approximation.

5.1 Run without prior information

First of all, we start FADSE with an initial randomly generated population, without prior information. We search for the optimal SLVP-based superscalar configurations considering the same two objectives, CPI and energy consumption, as in the previous manual DSE. We vary the parameters presented in Table 1 with the hope to find better configurations than our manually obtained ‘optimal’ configurations. To avoid extremes which can generate unfeasible configurations, we used the following constraints

$\text{UL2} > \text{DL1} + \text{IL1}$
$\text{UL2\_bsize} \ge \text{DL1\_bsize}$
$\text{UL2\_bsize} \ge \text{IL1\_bsize}$

where UL2_bsize, DL1_bsize and IL1_bsize are the block sizes for the unified $L2$ cache, $L1$ data cache and $L1$ instruction cache, respectively. In addition, we limited the cache sizes by using the following hard constraints (borders):

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

Unfortunately, the constraints used within the initial run do not allow FADSE to efficiently explore the borders and, therefore the configurations were not better than those obtained manually (see Fig. 2) from the energy point of view. Consequently, we relaxed the minimum cache capacities as follows:

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

As Fig. 2 shows, with relaxed borders FADSE provides significantly better configurations than our previous manual DSE. With these constraints the design space is reduced to 3% of the initial space, meaning $7.7*10^{13}$ (77 thousands of billions) configurations. Considering both objectives, the better results are influenced by the following parameters: less DL1 sets, less IL1 sets, higher decode/issue/commit width, higher ROB size, higher IQ size, higher number of MUL/DIV and higher SLVP size. Big structures will increase energy consumption thus we do need FADSE to find the relations between different parameters. We can observe again that SLVP helps in maintaining a better CPI and energy consumption when the cache sizes are reduced.

Since the exploration with relaxed borders was superior to the initial constraints, in the next experiment we used only the relaxed borders.

5.2 Run with manually obtained 'optimal' configurations

The second step in our experiment consists in starting FADSE with an initial randomly generated population but containing also our manually obtained 'optimal' configurations and their vicinity (with the goal to find better ones). We selected from Fig. 1 the best configuration in CPI, 2UL2_DL1, the best configuration in terms of energy consumption, 1/4UL2_1/8DL1, and other two configurations which are optimal from both CPI and energy viewpoints: 1/2UL2_1/2DL1 and 1/2UL2_1/4DL1. We also considered the vicinities of these four configurations by varying the SLVP size, $L1$ data cache size and $L2$ unified cache size one step up and down. Thus, we started FADSE again with randomly generated population but containing also our 24 selected configurations: the 'optimal' manual configurations and their vicinities (some of them are overlapped). Fig. 2 shows the obtained Pareto fronts after 25 generations by the first three runs (initial run, run with relaxed borders and run with initial good configurations) compared with the manually obtained configurations.

In terms of CPI all the runs find much better solutions than the manually obtained configurations. The run with relaxed borders clearly finds better configurations than the ones obtained through manual exploration and also better than the ones found during the initial run (restrictive constraints). The obtained solutions are distributed evenly along the Pareto approximated front.

Inserting good configurations into the initial population also provides good results but it is not able to explore the area with very low energies. It obtains better results than the run with relaxed borders in the vicinity of energy $1.20E+10$ [$W \cdot$ cycles]. Low energy configurations are not found probably because the initial configurations were better (and we have observed this on our analysis of the Pareto front approximation evolution over the generations) than all the other individuals inserted randomly in the population. So all these good individuals survived until the next generation and most of the offspring were generated from them thus loosing diversity. The mutation operator with a probability of 0.05 of changing one parameter has a small chance of influencing significantly the produced offspring, leading to a reduction of diversity.

5.3 Run with knowledge expressed through fuzzy rules

We are using fuzzy rules to allow the designer to express knowledge. The information provided by these fuzzy rules is then used during the search process to guide the DSE

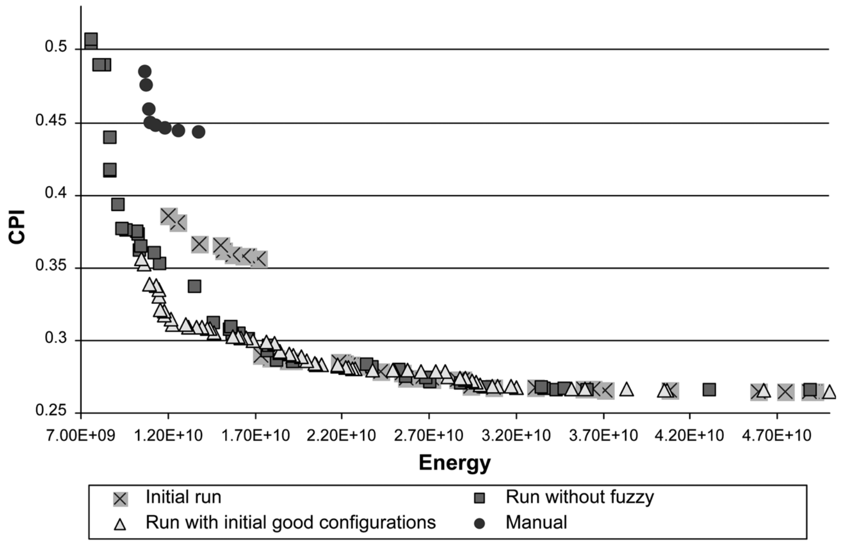

Fig. 2 *Pareto fronts comparison*

algorithm. For this purpose we have included the jFuzzyLogic library (http://jfuzzylogic.sourceforge.net) in FADSE. A user can define rules in a standard Fuzzy Control Language (FCL) file (IEC 61131 part 7). In this article, we have developed and implemented the following rules derived from our experience in computer architecture design:

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*

We have selected the Mamdani-type fuzzy systems [26]. These imply the following steps that need to be carried out to extract information: fuzzification of the input variables, evaluating the rules, aggregating the outputs and then defuzzification. The fuzzification was done using trapezoidal functions. For the evaluation the min function was used for 'and' and max for 'or'. For inference the Mamdani implication has used (min). The rule aggregation was performed using the Mamdani aggregation (max). This system was selected because of its popularity.

Two different mutation operators were used. Both of them are based on the bit flip mutation. To preserve diversity, the information provided by the fuzzy rules is not always taken into consideration. To obtain this, a probability of applying the fuzzy information (called fuzzy probability) is used. The only difference between the implemented methods is how the probability to apply the information provided by the fuzzy rules is computed.

In simple implementation this fuzzy probability is constant during the run of the algorithm and it is set to be equal with the probability of mutation (mutation_prob). If the fuzzy rule is not applied the algorithm switches to the classical bit flip mutation for the current parameter. The second implementation uses a Gaussian probability so there is a higher chance to apply the fuzzy rules for the first generation. As the DSE process runs, the probability to apply the fuzzy rules decreases to a value close to mutation_prob. We have selected the parameters of the Gaussian function such that at generation 5 the function is close to 0. The Gaussian function is then translated so that the minimum is close to mutation_prob. The final form of the function is shown below

$$f(x)_{\text{final}} = (1 - \text{mutation\_prob})\mathrm{e}^{(-(x)^2/2(150)^2)} + \text{mutation\_prob} \qquad (8)$$

where x increases with one for each individual generated by the algorithm. The result of this function is further multiplied by 0.8 and by the membership value [27] to obtain a maximum value less than 1.

Fig. 3 presents the results obtained with fuzzy information compared with the previous results.

We have selected the run with relaxed borders (called 'run without fuzzy') as the reference run since it has found solutions all along the approximated Pareto front. In Fig. 3 we compare the run with fuzzy information (constant probability to apply the fuzzy rules) with the run without fuzzy. It can be easily observed that the run with fuzzy information obtains very good results. Fig. 3 also shows that the run with fuzzy rules finds better results in the vicinity of energy $1.20\text{E}+10$ $[W \cdot \text{cycles}]$ than the run without fuzzy information. We have also compared the run with fuzzy rules with the one with initial good configurations and we observed that in fact the latter obtains a few individuals which are slightly better.

Fig. 4 performs a comparison between the results obtained with fuzzy information but with different methods of calculating the probability to use the information provided by them. The run with constant probability finds better individuals in the area with low energy. Having an almost 80% probability to apply the rules during the first generation might lead to a loss in diversity of the individuals on the parameters influenced by the rules. This fact might explain the poorer results.

Fig. 5 gives us two types of information: about the convergence of the algorithms and about the quality of results. It can be observed that the algorithms tend to stop rapid evolution after 15 generations (initial run is an exception). The algorithms were run until generation 25 owing to time constraints.

After a comparison of the hypervolume values we can conclude that: the initial run with restricted borders obtained the worst results. Relaxing the borders considerably improved the quality of results even though the size of the design space has become larger by a factor

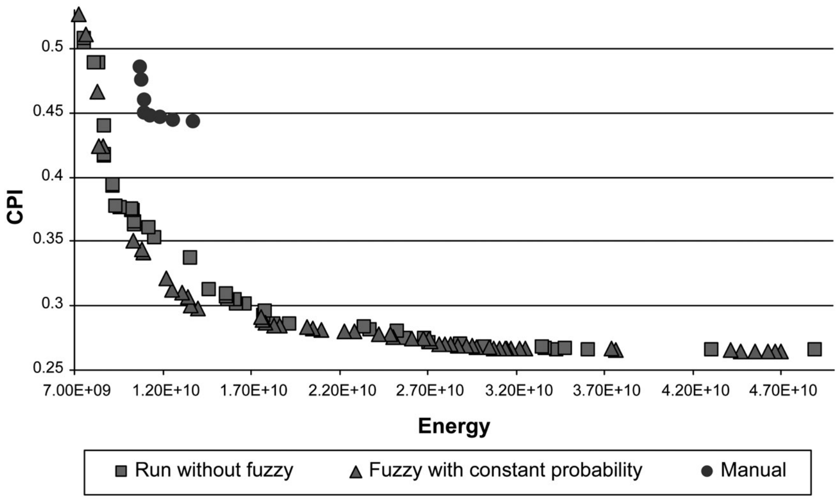

Fig. 3 *Pareto front comparisons between the run with fuzzy rules and the run with relaxed borders*

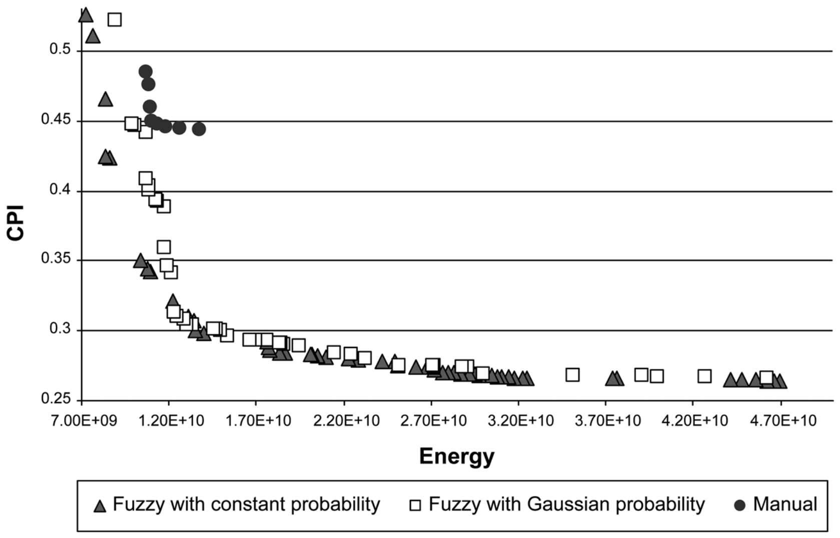

Fig. 4 *Pareto fronts comparison between the runs with fuzzy rules*

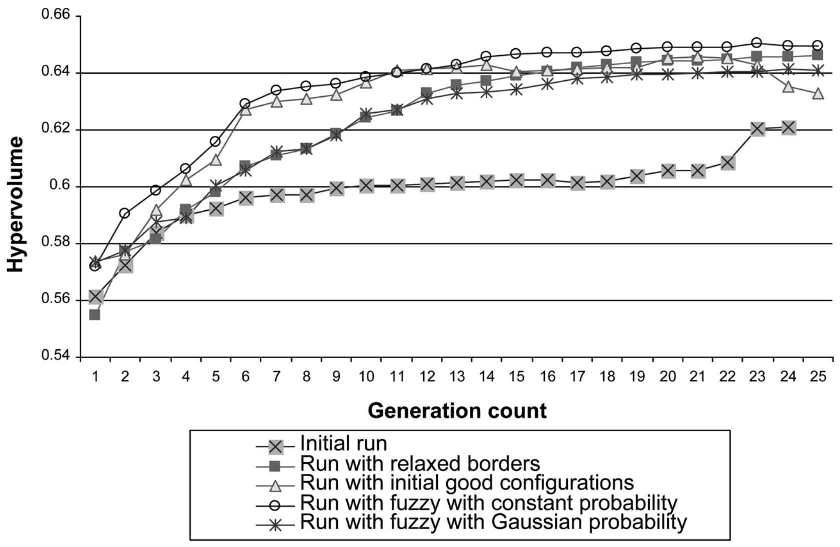

Fig. 5 *Hypervolume comparison*

of around 2 (from $3.8*10^{13}$ to $7.7*10^{13}$). Having good configurations inserted in the initial population can lead to very good results but it starts to perform poorly after generation 14, rising a bit after generation 19 and then dropping dramatically after generation 23. Observing the obtained Pareto front approximation and its evolution we have concluded that the algorithm tends to focus on a smaller area of the space, falling into local minima. This can be explained by the lack of diversity of the initial configurations, since all the individuals inserted differ on only two parameters from a total of 19.

The run with a constant probability of accepting the results from the fuzzy rules provided the best results. The run with a Gaussian probability of applying the information provided by the fuzzy rules had a similar behaviour at the beginning with the run using relaxed borders. After generation 12 the results are slightly worse. We can conclude that imposing a high probability of the rules will reduce diversity, especially with a small number of rules. In our previous work more rules were used and the membership functions had many intervals (associated linguistic terms) [28]. In this situation the runs with Gaussian probability provided better results.

It can be observed that using some extra knowledge (initial configurations or fuzzy rules) makes the algorithm start from a better initial population (see the hypervolume values at generation 1) and, as a consequence, the algorithm's convergence speed is better.

The hypervolume corresponding to the run with a constant probability of applying the fuzzy rules at generation 15, is reached by the run with relaxed borders only at generation 24. This is a great improvement. In our experiments, running one generation on 96 cores belonging to an Intel Xeon powered high performance computing (HPC) system, with cores running at 2 GHz, takes around one day. Running with fuzzy rules we achieved the same results 9 days earlier (36% faster) than without fuzzy rules. In addition, after the same amount of time (25 generations) the hypervolume reached by the fuzzy run is never reached by the simple run. If more qualitative information would have been provided through the fuzzy rules we do expect even bigger improvements. A single experiment takes around 25 days. Running all the five experiments took over 4 months of simulation on an HPC system using 96 cores.

All the runs evaluate roughly the same amount of individuals: around 2200 from the 2500 individuals sent for

Table 2 Parameter values for optimal configurations

Parameter		For high performance	For low energy
DL1 cache	sets	2048	32
	block size (bytes)	256	64
	associativity	2	2
IL1 cache	sets	1024	32
	block size (bytes)	16	256
	associativity	8	1
UL2 cache	sets	8192	256
	block size (bytes)	256	256
	associativity	4	16
SLVP (entries)		4096–8192	4096–8192
decode/issue/commit width		32/16/32	16–32/4–8/16–32
ROB/LSQ/IQ size (entries)		1024/512–1024/128	256/32–64/64
number of physical register sets (int/fp)		8/8	2/2
int/fp ALU		8/8	8/2
int/fp MUL/DIV		8/8	8/8

evaluation; the rest are reused from the database (12% reuse degree). This means that the produced offspring are almost all of them new/different individuals. This behaviour is caused by the extremely large design space. In previous explorations on different simulators (smaller design space – 10^6) around 60% reuse degree was observed [15, 16].

After analysing the best results obtained during the automatic DSE process we extracted the parameter values for optimal configurations from either high performance or low energy viewpoints. The results are presented in Table 2. As it can be observed, some parameters like Decode width and SLVP size must have high values in order to obtain both high performance and low energy.

6 Conclusions and further work

We have observed that the SLVP helps in maintaining a better CPI and energy consumption when the cache sizes are reduced. Therefore the optimal configurations, obtained by both the manual and automatic DSE of our SLVP-based superscalar architecture, have lower cache sizes than the baseline architecture without SLVP.

FADSE is able to find good configurations by evaluating a very small percentage of the total search space. We reduced the number of evaluated configurations to only 2500, representing $3*10^{-11}$% of the huge constrained design space of 77 thousands of billions configurations. The experimental results show that our automatic DSE provides significantly better configurations than our previous manual DSE.

Starting FADSE with initial good configurations can accelerate the DSE process. It is recommended that the configurations differ on multiple parameters so that diversity is preserved. In our situation the configurations differed only on three parameters thus leading to a loss of diversity and finally it could not explore the entire Pareto front.

Using fuzzy rules can considerably accelerate the DSE process (9 days earlier to reach the same result in our situation). Also, the obtained results are better than the ones obtained with no prior information after the same amount of time. In this concrete optimisation process the constant probability to apply the fuzzy rules lead to better results. In our previous work – where the number of rules was higher and had more linguistic terms associated with the membership functions – we have obtained better results by modulating the probability with a Gaussian function during the generations. With a larger number of rules the individuals are mutated into a more diverse population. Thus, forcing the rules to be applied often does not lead to a loss of diversity in the population.

We plan to repeat these experiments on SLVP-based SMT and multi-core architectures. Other further work possibilities are to access the SLVP only in the case of miss in both the *L*1 and *L*2 data caches, to index the SLVP table with the memory address instead of the instruction address, to exploit an N-value locality instead of 1-value locality as we are currently exploiting, to evaluate set-associative SLVP configurations and yet another one to design and implement an adaptive dynamic run-time thermal manager (temporarily deactivating the SLVP unit, voltage scaling, frequency scaling, migrating computation etc.). We also plan to explore, as the third objective, the die size needed by the memory architecture.

7 Acknowledgment

This work was partially supported by the POSDRU financing contract 7706 and by the Romanian National Council of Academic Research through the grant CNCSIS no. 485/2009-2011. Also, we would like to thank Camil Bancioiu for his support with the M-SIM connector and to Ciprian Radu for his useful comments.

8 References

1 Gellert, A., Florea, A., Vintan, L.: ‘Exploiting selective instruction reuse and value prediction in a superscalar architecture’, *J. Syst. Archit., Elsevier*, 2009, **55**, (3), pp. 188–195
2 Sharkey, J., Ponomarev, D., Ghose, K.: ‘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, 2005
3 Gellert, A., Palermo, G., Zaccaria, V., Florea, A., Vintan, L., Silvano, C.: ‘Energy-performance design space exploration in SMT architectures exploiting selective load value predictions’. Int. Conf. on Design, Automation and Test in Europe (DATE 2010), Dresden, Germany, March 2010, pp. 271–274
4 Calborean, H.: ‘Multi-objective optimization of advanced computer architectures using domain-knowledge’. PhD thesis, ‘Lucian Blaga’ University of Sibiu, Romania, 2011 (PhD Supervisor: Prof. Lucian Vintan, PhD), Sibiu, Romania, 2011
5 Lipasti, M.H., Wilkerson, C.B., Shen, J.P.: ‘Value locality and load value prediction’. Proc. Seventh Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, October 1996, pp. 138–147
6 Vintan, L., Florea, A., Gellert, A.: ‘Focalising dynamic value prediction to CPU’s context’, *IEE Proc. – Comput. Digit. Tech., Stevenage*, United Kingdom, 2005, **152**, (4), pp. 457–536
7 Chang, S.C., Li, W.Y.H., Kuo, Y.J., Chung, C.P.: ‘Early load: hiding load latency in deep pipeline processor’. Proc. Asia-Pacific Computer Systems Architecture Conf. (ACSAC), Taiwan, August 2008, pp. 1–8
8 Mutlu, O., Kim, H., Patt, Y.N.: ‘Address-value delta (AVD) prediction: a hardware technique for efficiently parallelizing dependent cache misses’, *IEEE Trans. Comput.*, 2006, **55**, (12), pp. 1491–1508
9 Sarangi, S.R., Liu, W., Torrellas, J., Zhou, Y.: ‘ReSlice: selective re-execution of long-retired misspeculated instructions using forward slicing’. Proc. 38th Annual IEEE/ACM Int. Symp. on Microarchitecture, Barcelona, Spain, 12–16 November 2005, pp. 257–270
10 Hilton A.D.: ‘Energy efficient load latency tolerance: single-thread performance for the multi-core era’. 2010, Publicly accessible Penn Dissertations. Paper 188, http://repository.upenn.edu/edissertations/188
11 Silvano, C., Fornaciari, W., Palermo, G., *et al.*: ‘MULTICUBE: multi-objective design space exploration of multi-core architectures’. Proc.

IEEE Int. Annual Symp. on VLSI, Lixouri Kefalonia, Greece, 2010, pp. 488–493

12 Desmet, V., Girbal, S., Temam, O., France, B.F.: 'Archexplorer. org: Joint compiler/hardware exploration for fair comparison of architectures'. Proc. Sixth HiPEAC Industrial Workshop, Paris, France, November 2008

13 Jia, Z.J., Pimentel, A.D., Thompson, M., Bautista, T., Núnez, A.: 'NASA: a generic infrastructure for system-level MP-SoC design space exploration'. Eight IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010, pp. 41–50

14 Kang, S., Kumar, R.: 'Magellan: a search and machine learning-based framework for fast multi-core design space exploration and optimization'. Proc. Conf. on Design, Automation and Test in Europe, Munich, Germany, 2008, pp. 1432–1437

15 Jahr, R., Ungerer, T., Calborean, H., Vintan, L.: 'Automatic multi-objective optimization of parameters for hardware and code optimizations'. Proc. Int. Conf. on High Performance Computing and Simulation, Istanbul, Turkey, July 2011, pp. 308–316

16 Calborean, H., Jahr, R., Ungerer, T., Vintan, L.: 'Optimizing a superscalar system using multi-objective design space exploration'. Proc. 18th Int. Conf. on Control Systems and Computer Science (CSCS), Bucharest, Romania, May 2011, pp. 339–346

17 Mariani, G., Palermo, G., Silvano, C., Zaccaria, V.: 'Meta-model assisted optimization for design space exploration of multi-processor systems-on-chip'. Proc. 12th Int. Euromicro Conf. on Digital System Design, Architectures, Methods and Tools. DSD'09, IEEE Computer Society, Patras, Greece, August 2009, pp. 383–389

18 Brooks, D., Tiwari, V., Martonosi, M.: 'Wattch: a framework for architectural-level power analysis and optimizations'. Proc. 27th Int. Symp. on Computer Architecture, Vancouver, Canada, June 2000, pp. 83–94

19 Shivakumar, P., Jouppi, N.P.: 'Cacti 3.0: an integrated timing, power, and area model'. WRL Research report, 2001, USA

20 Chen, J., Dubois, M., Stenström, P.: 'Integrating complete-system and user-level performance/power simulators: the simwattch approach'. Proc. IEEE Int. Symp. on Performance Analysis of Systems and Software, Austin, TX, USA, March 2003, pp. 1–10

21 Durillo, J.J., Nebro, A.J., Luna, F., Dorronsoro, B., Alba, E.: '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

22 Sierra, M.R., Coello, C.A.C.: 'Improving PSO-based multi-objective optimization using crowding, mutation and e-dominance' Evolutionary Multi-Criterion Optimization, (Springer-Verlag, 2005), pp. 505–519

23 Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: 'A fast and elitist multiobjective genetic algorithm: NSGA-II', *IEEE Trans. Evol. Comput.*, 2002, **6**, (2), pp. 182–197

24 Nebro, A., Durillo, J., Garcia-Nieto, J., Coello, C.A., Luna, F., Alba, E.: 'SMPSO: a new PSO-based metaheuristic for multi-objective optimization'. Proc. IEEE Symp. Series on Computational Intelligence, 2009, pp. 66–73

25 Zitzler, E., Laumanns, M., Thiele, L.: 'SPEA2: improving the strength Pareto evolutionary algorithm'. Eurogen, 2001, pp. 95–100

26 Mamdani, E.H., Assilian, S.: 'An experiment in linguistic synthesis with a fuzzy logic controller', *Int. J. Man-Mach. Stud.*, 1975, **7**, (1), pp. 1–15

27 Zadeh, L.A.: 'Fuzzy sets', *J. Inf. Control*, 1965, **8**, (3), pp. 338–353

28 Jahr, R., Calborean, H., Ungerer, T., Vintan, L.: 'Boosting design space explorations with existing or automatically learned knowledge'. Accepted for publication at 16th Int. GI/ITG Conf. on 'Measurement, Modelling and Evaluation of Computing Systems' and 'Dependability and Fault-Tolerance' (MMB & DFT 2012), Kaiserslautern, Germany, 19–21 March 2012