Multi-Objective Design Space Exploration Using Genetic Algorithms

Maurizio Palesi
Dip. Ing. Informatica e delle Telecomunicazioni
University of Catania
V.le Andrea Doria, 6
95125 Catania, Italy
mpalesi@diit.unict.it

Tony Givargis
Center for Embedded Computer Systems
UC Irvine, Info. & Comp. Sci.
444 Computer Science Building
Irvine, CA 92697-3425
givargis@ics.uci.edu

ABSTRACT

In this work, we provide a technique for efficiently exploring a parameterized system-on-a-chip (SoC) architecture to find all Pareto-optimal configurations in a multi-objective design space. Globally, our approach uses a parameter dependency model of our target parameterized SoC architecture to extensively prune non-optimal subspaces. Locally, our approach applies Genetic Algorithms (GAs) to discover Pareto-optimal configurations within the remaining design points. The computed Pareto-optimal configurations will represent the range of performance (e.g., timing and power) tradeoffs that are obtainable by adjusting parameter values for a fixed application that is mapped on the parameterized SoC architecture. We have successfully applied our technique to explore Pareto-optimal configurations for a number of applications mapped on a parameterized SoC architecture.

Keywords

Design space exploration, genetic algorithms, low power design, Pareto-optimal configurations, and system-on-a-chip architectures

1. INTRODUCTION

The growing demand for portable embedded computing devices is leading to new system-on-a-chip (SoC) architectures intended for embedded systems. Such SoC architectures must be general enough to be used across several different applications in order to be economically viable, leading to recent attention to parameterized SoC architectures. On the other hand, embedded computing devices, that are to be mapped onto these parameterized SoC architectures, often have very different design objectives such as different timing requirements or performance budgets. Therefore, parameterized SoC architectures must be optimally configured to meet varied timing requirements, power budgets, and, in general, multiple design objectives of a large class of applications. Consequently, there is a need for efficient multi-objective design space exploration approaches.

CODES'02. May 6-8, 2002, Estes Park, Colorado, USA.

A typical parameterized SoC architecture will have a processor core, one or more caches, on-chip bus hierarchy, on-chip memory, and a large number of peripheral cores that provide application specific functionality such as multi-media and communication processing. Each of these SoC cores is likely to be parameterized, enabling a designer to tune a core's settings for a specific application that is to be mapped on the parameterized SoC architecture. For example, the on-chip buses may be configured to use bus-invert [18] coding for low power, or the caches may be configured to use a greater or lesser degree of associativity for increased performance [13, 1]. An assignment of a value to each of these parameters will impact the overall timing, power, and other performance aspects of the system. Moreover, such performance impacts are highly dependent on the application running on the parameterized SoC architecture. Therefore, a designer must have a method for finding a feasible set of parameter values, referred to as a configuration of the parameterized SoC architecture, that meets the specification requirements.

We outline an exploration approach that efficiently searches the entire configuration space and outputs Pareto-optimal configurations providing the designer with only the interesting configurations that result in a tradeoff between the interesting design objectives (e.g., timing and power). Our approach augments the parameter dependency design space exploration technique previously established in [6] with a novel Genetic Algorithms (GAs) approach for improved performance.

The remainder of this paper is organized as follows. In Section 2, we define the problem and outline some background work. In Section 3, we give the GAs based design space exploration approach. In Section 4, we show our experimental results. In Section 5, we state our concluding remarks.

2. BACKGROUND

2.1 Problem Formulation

We are given a parameterized SoC architecture composed of numerous interconnected parameterized computational, communication, and memory elements. Each of these parameters can be assigned a value from a finite set of values. A complete assignment of values to all the parameters is a configuration. A complete collection of all possible configurations is the configuration space, (a.k.a., the design space). A partial collection of the configurations is a configuration subspace. We are also given a parameterized system-level model of the parameterized SoC architecture that when executed can yield multiple performance metrics (e.g., timing and

power) of the system under current configuration. Such parameterized simulation models have been outlined in [5, 17]. The problem is to efficiently compute, with the aid of a system-level model, the Pareto-optimal configurations, with respect to the performance metrics of interest, for a fixed application executing on the parameterized SoC architecture. For example, in the case of timing and power, a configuration is Pareto-optimal if no other configuration has better power for a given timing/speed.

Note that in general the solution to a multi-objective optimization problem can not be obtained by simply considering the design objectives separately. In practice, optimizing one design objective will adversely impact the optimality of other design objectives. The solution to such optimization problems falls under a class of strategies for multi-objective optimization. Multi-objective optimization (also called multi-criteria optimization, multi-performance or vector evaluation) can be defined as the problem of finding [4] a vector of decision variables (in our case a configuration vector that can be mapped on the parameterized system under study) which satisfies constraints and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria which are usually in conflict with each other. Hence, the term "optimize" means finding a solution which would give values for all the objective functions such as to be acceptable to the designer.

2.2 Design Space Exploration

The most straightforward but least efficient approach to determine the Pareto-optimal set of configurations of a parameterized SoC architecture with respect to a multi-objective design optimization criteria is to do an exhaustive search of the configuration space. This approach can be used only if the configuration space is limited. However, it is not rare to find parameterized SoC architectures with tens of parameters [7] and exponentially many configurations. Moreover estimating performance metrics for each configuration requires costly simulation and analysis of the system. For example in the target parameterized SoC architecture used in this paper, the evaluation of timing and power consumption given a single configuration requires on the average 1.5 sec. The configuration space of our experimental parameterized SoC architecture is of the order of 10^{12}, therefore an exhaustive search requires times of the order of tens of millions of years!

When the configuration space is too large to be explored in an exhaustive manner, heuristics must be used. One heuristic approach is to use evolutionistic techniques, such as GAs. GAs have found their way in many fields of VLSI design [14] at various levels of abstractions. For example, at the layout level, some partitioning [2], placement [16], and routing [12] techniques rely on GAs. At higher levels GAs have been used for power estimation [10], technology mapping [11], and netlist partitioning [3]. At even higher levels GAs have been used for reliable chip testing through efficient test vector generation [15]. Generally, the design space exploration problem as well as these other VLSI problems listed here are intractable (i.e., no polynomial time algorithm can guarantee optimal solution) and belong to either the NP-complete or NP-hard categories of problems. An approach based on GAs is very effective in solving such problems in a general and efficient way.

2.3 Genetic Algorithms

Evolutionary algorithms have been introduced by John Holland [9]. Since their introduction, a variety of evolutionary algorithms have been proposed [8]. The major ones are: GAs, evolutionary programming, evolution strategies, classifier systems, and genetic programming. They all share a common conceptual base of simulating the evolution of *individual* structures via processes of *selection*, *mutation*, and *reproduction*.

GAs are based on the evolution of a population of individuals over a number of generations. Each individual of the population is assigned a fitness value whose determination is problem dependent. At each generation, individuals are selected for reproduction based on their fitness value. Such individuals are crossed to generate new individuals, and the new individuals are mutated with some low mutation probability.

The objective of GAs is to find the optimal solution to a problem. However, because GAs are heuristics, the solution found is not always guarantied to be the optimal solution. Nevertheless, experience in applying GAs to a great deal of problems has shown that often the goodness of the solutions found by GAs is sufficiently high.

3. DESIGN EXPLORATION USING GA

We propose an approach for the exploration of the configuration space of a parameterized SoC architecture that uses GAs. More specifically, we have chosen a generic GA framework called SPEA2 [19], which, when applied to design space exploration, is very effective in finding points that are along the actual Pareto-optimal front. Our task is to map the design space exploration problem to this particular GA framework. We do this mapping as follows:

1. The representation of a configuration: a mapping between a possible configuration of the parameterized SoC architecture and a chromosome of the GA. Here, we use a gene for each parameter of the parameterized SoC architecture and allow that gene to assume only the values admissible by the parameter it represent.

2. The objective functions: a mapping between a configuration of the parameterized SoC architecture to a real value that is the measure of the performance metric that we want to optimize (e.g. timing, and power). In a multi-objective optimization criteria, we would have an objective function for each objective.

3. The convergence criterion: a criteria that determines when the evolution process of the GA should halt. One simple convergence criterion is to let the GA run for some fixed number of generations. While a simple criterion, it is not easy to determine the exact number of necessary iterations. To solve this problem we define a stop criterion based on distance convergence. The basic idea is to stop the evolution when there is no longer any appreciable improvement in the consecutive Pareto-optimal sets that are being found. The convergence criterion we propose uses a distance function [20] between two Pareto-optimal sets to establish when the GA has reached convergence. Let C' and C'' be two Pareto-optimal sets in the design space. The coverage function between C' and C'' is

$$f_C(C',C'') = \frac{|\{c'' \in C''; \exists\, c' \in C' : c' \preceq c''\}|}{|C''|} \qquad (1)$$

and represents the fraction of points in C'' that are dominated at least by one point in C'.

If C_i is the Pareto-optimal set at generation i and $G \geq 0$ then we define the following convergence index:

$$q(i,G) = f_C(C_{i+G}, C_i) - f_C(C_i, C_{i+G}) \quad (2)$$

As long as the evolutionary process improves the solutions found we continue to iterate (i.e., $f_C(C_{i+G}, C_i) \geq f_C(C_i, C_{i+G}) \Rightarrow q(i,G) \geq 0$). We could therefore perform an observation every G generations to evaluate $q(g,G)$ and determine if it has a value below some user-defined threshold T_c, which would signal the stop condition.

The main advantages of the use of GAs are are given below [14]:

- They are an adaptive approach in the sense that they are of general application and do not require detailed knowledge of the problem.
- They learn by experience, in the sense that they solve a problem by successive refinement.
- They are inherently parallel in the sense that at each iteration they evaluate not one but a number of possible solutions, equal to the size of the population.
- They are efficient at solving complex problems: this is demonstrated by the fact that evolutionary algorithms are currently receiving growing interest from researchers with various backgrounds to solve problems of all kinds and levels of complexity.
- The complexity of the approach often lies in evaluation of the fitness functions of the individuals in each generation. This procedure can be parallelised quite simply, as it is possible to assign individual fitness values independently, so concurrent execution of this operation does not cause conflict.

3.1 Exploration Algorithm

Previously, it has been shown that by taking parameter interdependencies into account, the design space can be extensively pruned [6]. Such parameter dependency awareness is deployed in a design exploration tool called Platune [6]. Platune works in two phases. In the first phase the design subspace defined by clusters of interdependent parameters are explored in an exhaustive manner to find the local Pareto-optimal set (LPOS). In the second phase these LPOS are merged and exhaustively searched to find the global Pareto-optimal set (POS). Platune works well as long as most of the parameters are not interdependent, as this will result in a large number of small clusters that can be feasibly searched in an exhaustive manner. But if the parameters are heavily interdependent, the approach in Platune becomes infeasible. In this work, we substitute a GA based approach in place of the exhaustive search used by Platune when the subspace to be searched is greater in size than some threshold T. Thus, our approach is a merger of the parameter dependency approach introduced in Platune with a GA search introduced in this work.

The new algorithm works as follows. In the first phase, if the size of the configuration space C generated by a cluster is greater than T we apply GAs on that subspace, initializing the population with random configurations from C. Else we do an exhaustive search as shown in Algorithm 1.

Algorithm 1 GAPlatune: an iteration of the 1st phase

Require: C: set of configurations to be explored
Ensure: $LPOS$: local Pareto-optimal set of the cluster C
  if $|C| > T$ **then**
    GA.InitRandomPopulation(C);
    $LPOS$ = GA.Evolve();
  else
    $LPOS$ = ExhaustiveSearch(C);
  end if

In the second phase, given two clusters C_i and C_j and the respective LPOSs ($LPOS_i$ and $LPOS_j$), if the size of the configuration space generated by merging $LPOS_i$ and $LPOS_j$ is greater than T then the GA is applied. In this case, differently from the first phase, the initial population of the GA is initialized with the configurations from $LPOS_i$ and $LPOS_j$. Else if the size of the configuration space is less than T an exhaustive search is applied, as shown in Algorithm 2.

Algorithm 2 GAPlatune: an iteration of the 2nd phase

Require: $LPOS_i, LPOS_j$: Pareto-optimal set of the clusters C_i and C_j
Ensure: $LPOS_{ij}$: local Pareto-optimal set obtained by merging the clusters C_i and C_j
  $C = LPOS_i \times LPOS_j$;
  if $|C| > T$ **then**
    GA.InitPopulation($LPOS_i, LPOS_j$);
    $LPOS_{ij}$ = GA.Evolve();
  else
    $LPOS_{ij}$ = ExhaustiveSearch(C);
  end if

3.2 Algorithm Evaluation

To evaluate the quality of the obtained results we define a goodness index as being the average distance (as a percentage) between the approximated Pareto-optimal set ($\mathcal{A}$) obtained using the mixed approach described above and the actual Pareto-optimal set (O) obtained by performing exhaustive only searches as done in Platune [6]. For the following discussion, and without loss of generality, we assume that the metrics of interest are timing (i.e., execution time) and power.

Let $\mathcal{A}$ and O be ordered sets of power, and timing pairs, sorted in increasing values of power. As the power and timing values are on different scales, the components of each point in the set $O \cup \mathcal{A}$ are normalized to the maximum power and timing values, thus obtaining the normalized sets O_n ed $\mathcal{A}_n$, as shown in Figure 1(a) and (b).

Let $d(T, O_n)$ be the distance between a point $T \in \mathcal{A}_n$ and the polyline generated by O_n. This distance is 0 if T is not dominated by any point in $\mathcal{A}_n$. If, on the other hand, T is dominated by at least one point in O_n then $\mathcal{S}(T) \subseteq O_n$ is the set of the pairs (Q_i, Q_{i+1}) such that the angles α and β respectively formed by the lines passing through T and Q_i and through T and Q_{i+1} with the line joining Q_i and Q_{i+1} (see Figure 2(a)) is less than 90 degrees.

If $\mathcal{S} \neq \emptyset$ then $d(T, O_n)$ is the minimum distance between T and the segments defined in $\mathcal{S}$:

$$d(T, O_n) = \min\{d_s(T,S) : S \in \mathcal{S}\}$$

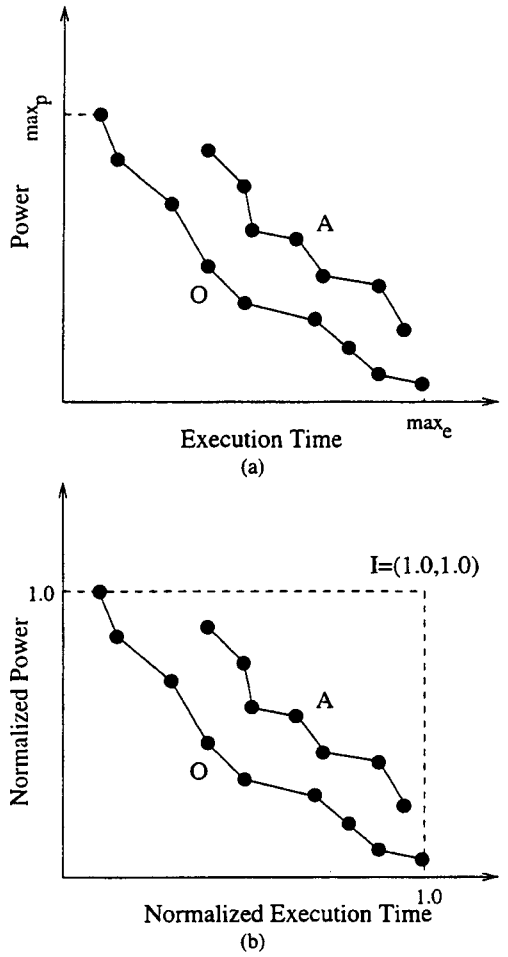

Figure 1: Pareto-optimal set and approximated pareto-optimal set; (a) before normalization, (b) after normalization.

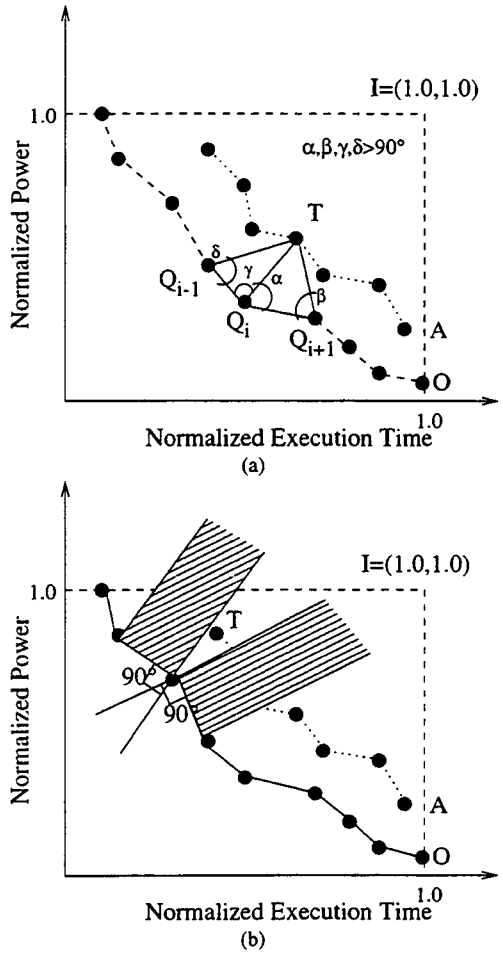

Figure 2: Distance between a point and a polyline. (a) The angles α, β, γ, and δ are less than 90 degrees, so $S(T) = \{(Q_{i-1}, Q_i), (Q_i, Q_{i+1})\}$. (b) There is no line passing through T and crossing in a perpendicular direction a segment of the polyline O.

where the function $d_s(T,S)$ returns the distance between a point T and the line passing between the two points in S.

If $S = \emptyset$ (as in Figure 2(b)) then $d(T, O_n)$ is the minimum Euclidean distance between T and each point in O_n:

$$d(T, O_n) = \min\{d_p(T,Q) : Q \in O_n\}$$

where the function $d_p(T,Q)$ returns the Euclidean distance between the point T and the point Q.

Having defined the distance between each point in $\mathcal{A}_n$ and O_n, we can define the average distance between the sets O_n and $\mathcal{A}_n$:

$$d_m(O_n, \mathcal{A}_n) = \frac{1}{|\mathcal{A}_n|} \sum_{T \in \mathcal{A}_n} d(T, O_n)$$

To have an idea of the percentage difference between A and O, we have to relate $d(O_n, \mathcal{A}_n)$ to the maximum distance between the point $I = (1.0, 1.0)$ and the points in the set $O_n \cup \mathcal{A}_n$.

$$d_{\%}(O_n, \mathcal{A}_n) = 100 \times \frac{d_m(O_n, \mathcal{A}_n)}{\max\{d_p(I,T) : T \in O \cup \mathcal{A}\}} \quad (3)$$

We use the evaluation technique outlined here in the next section to evaluate the quality of our exploration approach.

4. EXPERIMENTS

We have applied both the dependency/exhaustive approach, used by Platune, (DA) and the mixed approach, presented in this work, (MA) to a highly parameterized SoC architecture shown in Figure 3. Our target architecture consists of a MIPS R3000 processor, an instruction cache (I$), a data cache (D$), on-chip memory, and various busses connecting the CPU and the caches as well as the caches and the on-chip memory. Each component of this architecture is parameterized as shown in Figure 3. Note that these parameters refer to architectural or micro-architectural features and are

technology independent. There are a total of 19 parameters generating a space of 5.97×10^{12} possible configurations.

The methodology proposed has been validated in terms of both the quality of the solutions found and efficiency of execution. The index used to measure the quality of the solutions is the average distance (as a percentage) of the approximated Pareto-optimal set (APOS) found by MA from the POS found by DA (using Equation 3). Efficiency is measured by counting the number of simulations required to complete the exploration.

Exploration of the configuration space is confined to the subspace obtained by fixing the *voltage scale* parameter, as voltage scaling is usually performed dynamically. For each benchmark the exploration is performed using both DA and MA. In the latter case the internal and external population is set as comprising of 50 individuals, using a crossover probability of 0.9 and a mutation probability of 0.01. With regard to the convergence criterion, the term G in condition 2 is set to 3 while the convergence threshold T_c is set to 0.05. Four tests are carried out for each benchmark with four different threshold values: T=100, 200, 400 and 800.

Table 1 shows the results obtained by running a set of benchmarks from the Motorola PowerStone suite (a collection of embedded and portable applications) for both DA and MA approaches. The first column states the benchmark name. The second column states the CPU time required to evaluate a configuration of the system when it executes that application (etime). This time has been measured using the unix command `time` on a Athlon 800 MHz workstation with 256 MB of RAM running Linux. The third column shows the number of configuration visited using the DA to find the POS (evals). The remaining columns refers to the results obtained using the MA for four different thresholds (T=100, 200, 400 and 800). For each threshold the three columns represent the number of configurations visited to extract the APOS, the average distance (as a percentage) of the APOS from the POS (d%), and the percent saving in terms of the simulation time with respect to DA (s%). From the efficiency point of view, on average, we obtain 80% savings in the number of simulations. On the other hand the average distance from the POS is less than 1% for all threshold values. The last line in the table gives the arithmetical average of the values in each column.

Figure 4 shows the cumulative distribution of the average distances of the APOS from the POS for different threshold values. For each threshold over 95% of the APOS have a distance less than 1% from the POS. In particular lower thresolds (such as T=100 and T=200) give better results than higher thresholds (T=400 and T=800).

To summarize, experiments with a number of benchmarks show that using the GAs we obtain a Pareto-optimal set that is within 1% of the actual set but with less effort, namely, an 80% reduction in simulation time.

5. CONCLUSION

We have outlined an approach that uses GAs to improve the performance of existing design space exploration algorithms that seek to find Pareto-optimal configurations of parameterized SoC architectures while taking into account multiple design objectives. Specifically, our approach replaces the exhaustive component of the parameter interdependency based approach called Platune [6] by replacing it with a technique that is based on a GA framework called SPEA2 [19]. Experiments show that on the average a saving of

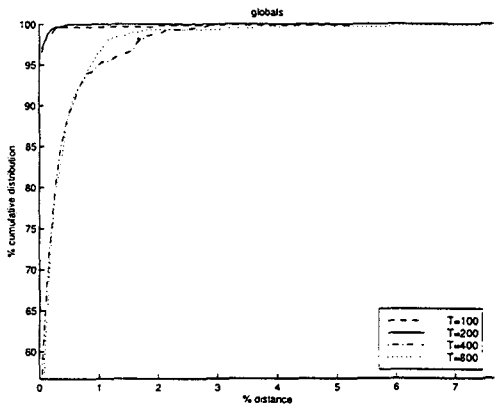

Figure 4: Percent cumulative distribution of the distances between APOS and POS for different threshold values.

80% in simulation time is achievable while still maintaining exploration results that are within 1% of those generated by an exhaustive approach.

6. REFERENCES

[1] D. Albonesi. Selective cache ways: On-demand cache resource allocation. *MICRO*, 1999.

[2] C. J. Alpert, L. W. Hagen, and A. B. Kahng. A hybrid multilevel/genetic approach for circuit partitioning. In *Fifth ACM/SIGDA Physical Design Workshop*, pages 100–105, Apr. 1996.

[3] C. J. Alpert and A. B. Kahng. Recent developments in netlist partitioning: A survey. *VLSI Journal*, 19(1–2):1–81, 1995.

[4] C. A. C. Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, Aug. 1999.

[5] T. Givargis, F. Vahid, and J. Henkel. Fast cache and bus power estimation for parametrized system-on-a-chip design. In *Design Automation and Test in Europe (DATE)*, Mar. 2000.

[6] T. Givargis, F. Vahid, and J. Henkel. System-level exploration for pareto-optimal configurations in parameterized systems-on-a-chip. In *International Conference on Computer Aided Design*, Nov. 2001.

[7] R. E. Gonzalez. Xtensa: A configurable and extensible processor. *IEEE Micro*, pages 60–70, 2000.

[8] J. Heitkötter and D. Beasley. The hitch-hiker's guide to evolutionary computation. `http://surf.de.uu.net/encore/www/`, Apr. 12 2001.

[9] J. H. Holland. *Adaption in Natural and Artificial Systems*. MI: University of Michigan Press, 1975.

[10] Y.-M. Jiang, K.-T. Cheng, and A. Krstic. Estimation of maximum power and instantaneous current using a genetic algorithm. In *Proceedings of IEEE Custom Integrated Circuits Conference*, pages 135–138, May 1997.

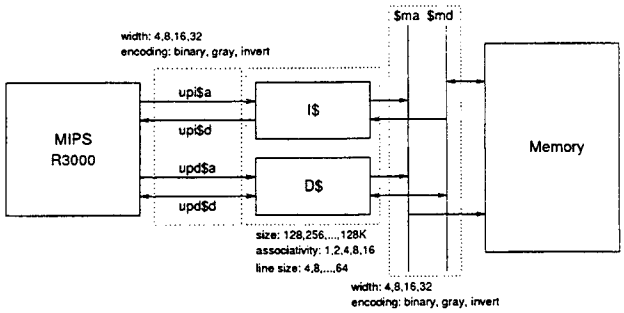

Figure 3: Reference architecture and its parameters.

Benchmark	etime (s)	DA	MA											
			T=100			T=200			T=400			T=800		
		evals	evals	d%	s%	evals	d%	s%	evals	d%	s%	evals	d%	s%
g3fax	2.61	27382	3052	0.01	88.9	2657	0.02	90.3	3778	0.00	86.2	3712	0.0615300	86.4
jpeg	12.22	15996	2481	0.11	84.5	2801	0.01	82.5	2855	0.01	82.2	2912	0.1904200	81.8
qurt	0.09	13231	2359	0.02	82.2	2377	0.00	82.0	2967	0.00	77.6	4183	0.1657300	68.4
bcnt	0.13	22642	2318	0.01	89.8	3183	0.01	85.9	2338	0.01	89.7	4276	0.1532600	81.1
adpcm	0.34	14694	2535	0.01	82.7	3143	0.01	78.6	3122	0.00	78.8	3193	0.3672900	78.3
blit	0.17	37787	2928	0.02	92.3	3254	0.01	91.4	3406	0.00	91.0	2979	0.0960800	92.1
compress	0.54	14035	2607	0.01	81.4	3237	0.02	76.9	3475	0.00	75.2	3373	0.8247200	76.0
crc	0.2	21205	3577	0.01	83.1	3503	0.01	83.5	3446	0.00	83.7	2517	0.2054900	88.1
des	0.18	21970	2513	0.01	88.6	2143	0.02	90.2	3822	0.01	82.6	3933	0.2510100	82.1
engine	1.18	15532	2910	0.03	81.3	2655	0.01	82.9	3349	0.00	78.4	3711	0.1542900	76.1
fir	0.13	16832	3150	0.05	81.3	3659	0.01	78.3	2929	0.00	82.6	3210	0.2118100	80.9
pocsag	0.68	19102	2228	0.02	88.3	2807	0.00	85.3	3070	0.00	83.9	3388	0.1118200	82.3
ucbqsort	0.65	19102	2830	0.01	85.2	3103	0.01	83.8	2942	0.00	84.6	2801	0.1502500	85.3
Average	1.47	19962	2730	0.02	85.3	2963	0.01	84.0	3192	0.00	82.8	3399	0.2264385	81.5

Table 1: Average distance between APOS and POS and a saving in terms of simulation time between MA and DA for different thresholds.

[11] V. Kommu and I. Pomenraz. GAFAP: Genetic algorithm for FPGA technology mapping. In *European Design Automation Conference*, pages 300–305, 1993.

[12] J. Lienig and K. Thulasiraman. A genetic algorithm for channel routing in VLSI circuits. *Evolutionary Computation*, 1(4):293–311, 1993.

[13] A. Malik, B. Moyer, and D. Cermak. A programmable unified cache architecture for embedded applications. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2000.

[14] P. Mazumder and E. M. Rudnick. *Genetic Algorithms for VLSI Design, Layout & Test Automation*. Prentice Hall, Inc., 1999.

[15] D. Saab, Y. Saab, and J. Abraham. Automatic test vector cultivation for sequential vlsi circuits using genetic algorithms. *IEEE Transactionsn Computer-Aided Design*, 15(10):1278–1285, Oct. 1996.

[16] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using metagenetic parameter optimization. *IEEE Transactions on Computer-Aided Design*, 9:500–511, May 1990.

[17] T. Simunic, L. Benini, and G. D. Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In . ACM Press, editor, *Proceedings of the 36th Conference on Design Automation*, pages 867–872, New Orleans, LA, USA, June21–25 1999.

[18] M. R. Stan and W. P. Burleson. Bus invert coding for low power I/O. *IEEE Transactions on VLSI Systems*, pages 49–58, Mar. 1995.

[19] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the performance of the strength pareto evolutionary algorithm. Technical Report TIK-Report 103, Computer Engineering and Communication Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092, May 2001.

[20] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 4(3):257–271, Nov. 1999.