Parameterised System Design Based on Genetic Algorithms

Giuseppe Ascia
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania
V.le Andrea Doria, 6
95125 Catania - Italy
gascia@diit.unict.it

Vincenzo Catania
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania
V.le Andrea Doria, 6
95125 Catania - Italy
vcatania@diit.unict.it

Maurizio Palesi
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania
V.le Andrea Doria, 6
95125 Catania - Italy
mpalesi@diit.unict.it

ABSTRACT

A recent reduction in the time to market has led to the development of a new approach to IP-based design in which a highly parametric pre-designed system-on-a-chip is configured according to the application it will have to execute. The greatest problems in this area regard exploration of the range of possible system configurations in search of the optimal configuration for a given system. There are, in fact, a number of parameters involved (bus sizes, cache configurations, software algorithms, etc.), each of which has a great impact on design constraints such as area, power and performance. An exhaustive analysis of all possible configurations is thus computationally unfeasible. In this paper we propose using genetic algorithms to determine the optimal configuration for a highly parametric system. The approach is applied to the search for the optimal configuration (in terms of area, power and mean access time) of a memory hierarchy involved in a given application.

Keywords

Parameterised systems, Genetic algorithms, Exploration of system configurations

1. INTRODUCTION

A reduction in the time to market has led to the definition of a new approach to IP-based design called *configure-and-execute* [23]. It is based on the presence of highly parametric IPs (Intellectual Properties) representing the basic components of a SoC (System-on-a-Chip). Once the architecture of a system has been designed, that is, it has been decided which IPs to use, it is necessary to find the optimal configuration for them according to the specific application (or set of applications) that have to be executed. The chosen values for these parameters (bus sizes, coding teqniques, cache parameters, arbitration schemes, etc.) are the ones that optimise a function that almost always depends on three main variables: area, power and performance.

There are two main problems to be dealt with here. The first one regards the definition of the function to be optmised, the second point is which strategy to use in the search for the optimal parameter configuration. Evaluation of the objective function for a given configuration requires simulation of a system model, which may be of a varying degree of computational complexity according to the level of detail required. Fortunately there has recently been a spread in the use of general-purpose programming languages in system modelling (especially C/C++) to obtain what are called *executable specifications* [22]. One of the main advantages of this approach is that it provides a model of the whole system right from the earliest design stages. This enables the designer to simulate the whole system much faster than would be possible with a description in HDL (VHDL or Verilog) because it works at a higher level of abstraction. In addition, the opportunity to integrate a high-level model with estimation engines provides information about variables (e.g. area, power etc.), typically obtained at a lower level, with a sufficient degree of accuracy [14].

These models, which are designed to be highly parametric, can be used to simulate a very large number of possible configurations [13]. Unfortunately, the vast range of configurations often makes an exhaustive search for the optimal configuration computationally unfeasible [12]. In [7] a hybrid approach using evolutionary algorithms and heuristic techniques was used to optimise the mapping of an algorithm-level specification onto a heterogeneous hardware/software architecture. This approach, which follows the *describe-and-synthesize* design paradigm, is different from the one we intend to follow, known as *configure-and-execute*, the advantages of which are fully discussed in [23].

In this paper we present a strategy to search for the optimal configuration of a parameterisable system using genetic algorithms. The definition of new methods is, in fact, of fundamental importance, above all in the field of SoCs, where the same application is executed throughout the system's lifetime. As a case study we will use the optimisation, in terms of area, power consumption and performance, of a memory hierarchy.

CODES 01 Copenhagen Denmark

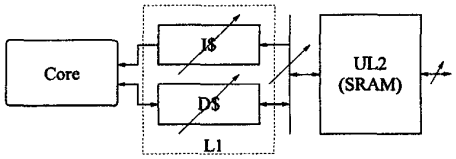

Figure 1: Reference architecture.

The paper is structured as follows. Section 2 describes the parametric reference architecture and the estimation flow of the variables to be optimised. Section 3 highlights the necessity to find a trade-off function. Section 4 presents our proposal to apply genetic algorithms to an efficient search for the optimal configuration. Section 5 presents a case study. Finally, Section 6 provides our conclusions and indications as to future developments.

2. REFERENCE ARCHITECURE

Figure 1 shows the architecture to which reference will be made. It consists of a memory hierarchy in which the first two levels are on-chip [3] and the third is the external memory. The first-level cache is separate for instructions and data, while the second-level is unified. The parts of the graph crossed by an arrow represent the parametric elements and are the caches and data buses. The caches can be configured in relation to size, block size, associativity, the replacement policy (lru, fifo, random) and, in the case of data caches, the writing policy (write through, write back) and the way writing misses are handled (write allocate, no-write allocate). The data buses can be configured in terms of the number of lines.

Henceforward the term *memory hierarchy configuration* (MH) will be taken to mean the set of cache (C) and bus (B) configurations.

$$MH = \{C, B\}$$

By *cache configurations* we mean the configuration of the first-level instruction cache (C_{IL1}), the first-level data cache (C_{DL1}) and the unified second-level cache (C_{UL2}).

$$C = \{C_{IL1}, C_{DL1}, C_{UL2}\}$$

By *cache configuration* we mean the following set of parameters: cache size (S), block size (BS), associativity (A), replacement policy (RP) and, in the case of data caches, the writing policy (WP) and writing misses policy (WMP).

$$C_{IL1} = \{S_{IL1}, BS_{IL1}, A_{IL1}, RP_{IL1}\}$$
$$C_{DL1} = \{S_{DL1}, BS_{DL1}, A_{DL1}, RP_{DL1}, WP_{DL1}, WMP_{DL1}\}$$
$$C_{UL2} = \{S_{UL2}, BS_{UL2}, A_{UL2}, RP_{UL2}, WP_{UL2}, WMP_{UL2}\}$$

By *bus configurations* we mean the size of the bus between the first and second levels ($B_{L1 \leftrightarrow L2}$) and between the second level and the external memory ($B_{L2 \leftrightarrow EM}$).

$$B = \{B_{L1 \leftrightarrow L2}, B_{L2 \leftrightarrow EM}\}$$

The decision to focus on a subset of a complex system (i.e. only the memory hierarchy) is accounted for by the strength of the impact of its configuration, in terms of area, performance and power consumption.

The impact on area is considerable as a consistent portion of silicon is devoted to the caches and their control logic (see the photos of the dies of current processors in which 15%-40% of the die area is occupied by the cache [1]).

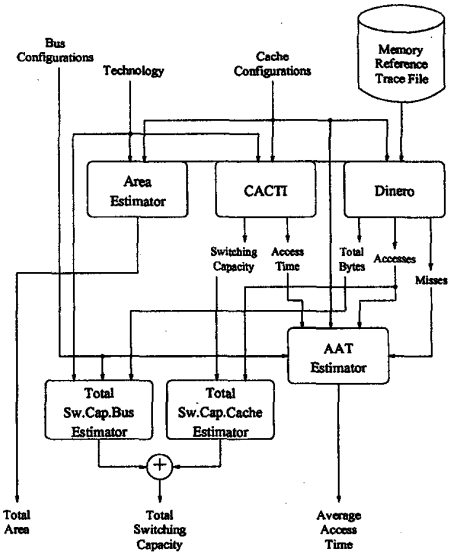

Figure 2: Estimation flow.

The overall performance of a system greatly depends on the memory hierarchy which exploits the principle of reference locations to hide the latency of a slow memory from the processor [16]. As the memory is accessed each clock cycle (to fetch instructions) and the most frequent operations are load/store, the impact of the configuration of the memory hierarchy on performance is immediately obvious [17, 10].

Power dissipation also depends on the memory hierarchy [4]. The amount of energy spent on communications between the processor and the memory is known to represent a significant portion of the total absorption (in fact information coding techniques have been defined to minimise the activity induced by data on high-capacitance buses [18]). A well-designed memory hierarchy limits data traffic on buses outside the chip (and thus of high capacitance), thus reducing the total amount of power absorbed.

2.1 Estimation Flow

Figure 2 shows the structure of the estimation framework used. The input is represented by the configuration of the memory hierarchy, specification of the technology used (there are two: .35μm and .80μm) and a memory reference trace file. The output is an estimate of the amount of area occupied by the memory hierarchy, the average memory access time for the processor and the total switching capacitance. The average length of time required for the processor to access the memory will be the yardstick whereby we will measure performance: the shorter it is the better the performance of the system will be. The total switching capacitance will be the yardstick used to measure power consumption, to which it is proportional.

The memory reference trace files used contain 1,000,000 references for each benchmark in the SPEC92 suite [2] and

were generated on an MIPS architecture. Each benchmark, together with the configuration of the memory hierarchy, represents the input to the cache simulator *Dinero* [6] which enables us to determine the number of accesses and misses for each cache and the total number of bytes transferred from one level in the hierarchy to the next.

Having established the configuration of a cache and the technology, we used *CACTI* [20] to estimate the access time and the switching capacitance per access. To estimate the area we used the model proposed in [19] which gives the area occupied in terms of *register bit equivalent* (rbe). To convert from *rbe* to mm^2 we used the data in [9].

The capacitance of each line of the bus connecting the first level to the second depends on the number of lines. The area required for bus routing was taken as fixed. If the length is the same, the capacitance of a line of a bus whose width is n will be lower than that of a bus whose width is $m > n$, as they will be closer and so the effects of the coupling capacitance will be greater [12]. To determine the capacitance of a line of a bus of a certain size we used the equation defined in [11]. To determine the bus's switching capacitance random traffic was considered.

To determine the average access time (AAT) required for the processor to access the memory, we used the classic sequential forward model. The data item required by the processor is sought in the first level. If it is not found (miss) it is sought in the second level and so on until it is found (hit). At this point the data is transported back through the hierarchy and delivered to the processor. If the memory hierarchy has N levels we have:

$$AAT = AT_1 + MR_1 \cdot (HR_2 \cdot MP_2 + + MR_2 \cdot (\ldots + MR_{N-1} \cdot MP_N)\ldots) \quad (1)$$

where AT_1 is the access time for the first-level cache (i.e. the one directly connected to the processor). MR_n is the local miss rate for level n , i.e. the ratio between the number of misses at level n and the total number of accesses at that level. HR_n is the local hit rate for level n ($HR_n = 1 - MR_n$). MP_n is the time required to transport the data required from level n to level 1:

$$MP_n = \sum_{i=n}^{2} TT_i = \sum_{i=n}^{2} \left(AT_i \frac{BS_{i-1} \times 8}{BL_{i-1}} \right) = MP_{n-1} + TT_n$$

where TT_i is the time required to transfer a block from level i to level $i-1$, which obviously depends on the number of lines for the bus connecting level i and level $i-1$. AT_i is the access time for the cache at level i. BS_i is the size of the block (in bytes) for the cache at level i. BL_i is the number of lines for the bus connecting level i to level $i+1$.

Applying equation 1 to our reference architecture, we have:

$$AAT_{inst} = AT_{il1} + MR_{il1} \cdot HR_{ul2} \cdot MP_{ul2} + + MR_{il1} \cdot MR_{ul2} \cdot MP_{mm}$$

$$AAT_{data} = AT_{dl1} + MR_{dl1} \cdot HR_{ul2} \cdot MP_{ul2} + + MR_{dl1} \cdot MR_{ul2} \cdot MP_{mm}$$

which respectively represent the average access times required for the CPU to access the instructions and data caches. The average access time used to measure performance was calculated as the average of AAT_{inst} and AAT_{data} weighted by the number of accesses to the instructions and data caches respectively.

$$AAT = \frac{AAT_{inst} \times Acc_{inst} + AAT_{data} \times Acc_{data}}{Acc_{inst} + Acc_{data}} \quad (2)$$

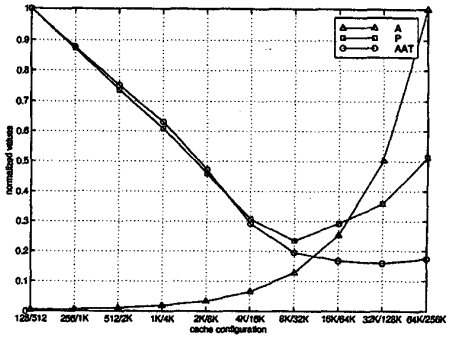

Figure 3: Normalised values of area, power and average access time with various cache size.

3. TRADE-OFFS

In this section we will describe the effects of varying the configuration of the memory hierarchy on area, performance and power consumption. Figure 3 shows the normalised trends for area, total switching capacitance and average access time with various cache sizes. In each configuration the first-level caches are of the same size while the second-level cache is 4 times the size of the first-level one. The trend followed by the area is a predictable one: as the cache size increases, so does the area. The total switching capacitance takes a less obvious trend: it is maximum with the first configuration, then decreases down to the configuration 8K/8K/32K and then starts to increase again. This behaviour can be accounted for by the fact that in the configurations before 8K/8K/32K the first-level cache is unable to capture the working-set of the benchmark, causing an excessive number of accesses to the next levels and thus greater power consumption. With the configurations after 8K/8K/32K, even though the first-level caches capture almost all the CPU accesses, as they are of a large size they consume more power. The average access time decreases rapidly up to the configuration 4K/4K/16K, then slows down as far as 32K/32K/128K and then starts to grow again. This can be accounted for as follows: as the size of the caches increases, there is on the one hand a loss due to an increase in the latency for the first-level caches, and on the other a gain because there are fewer accesses to the next levels which feature much greater latency.

The choice of the configuration for the memory hierarchy depends on the objective to be optimised. To sum up, three different configurations optimising three different objectives were found: if the optimisation is to be in terms of area, the configuration to choose is 128/128/512; if only power consumption is of interest then the optimal choice is 8K/8K/32K; if, on the other hand, the aim is to optimise performance, then the best configuration is 32K/32K/128K. As said previously, at times it is desirable to optimise in more than one direction. The design objectives often require the optimisation of several contrasting variables: an

enhancement of one means a deterioration of another and vice versa. As can be seen (Fig. 3), in fact, performance improvements lead to an increase in area and power consumption. For this reason the choose of a configuration requires definition of a trade-off function.

4. METHODOLOGY PROPOSED

An exhaustive search for the optimal configuration of a parameterisable system is computationally unfeasible. We can take our reference architecture as an example and calculate the range of possible configurations. Let us assume that the range of variation for the parameters involved is as follows:

$S_{IL1}, S_{DL1} \in \{128, 256, 512, 1K, 2K, 4K, 8K, 16K, 32K\}$
$S_{UL2} \in \{1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, 512K, 1M, 2M\}$
$BS_{IL1}, BS_{DL1} \in \{8, 16, 32\}$
$BS_{UL2} \in \{8, 16, 32, 64, 128\}$
$A_{IL1}, A_{DL1} \in \{1, 2, 4, 8\}$
$A_{UL2} \in \{1, 2, 4, 8, 16\}$
$RP_{IL1}, RP_{DL1}, RP_{UL2} \in \{lru, fifo, random\}$
$B_{L1 \leftrightarrow L2} \in \{32, 64, 128, 256\}$

If we exclude inadmissible configurations (first-level cache/block size greater than second-level), the number of possible configurations is 213,062,400. In our analysis flow, analysis of a configuration with a benchmark of 1,000,000 references requires about 1 sec. on a 450MHz UltraSparcII workstation. Once the benchmark is set, an exhaustive search for the best configuration would take nearly 7 years' simulation!.

In this section we will propose a methodology for exploring the range of configurations using genetic algorithms. Genetic algorithms make it possible to find solutions to problems for which the solution method is unknown: it is sufficient to define a fitness function (in our case the cost function to be minimised) and leave the algorithm to converge on the optimal configuration.

The tool based on genetic search techniques we used was GENESIS [15]. The genome of each individual in the population represents a possible configuration. In our case it is a 34-bit structure: the sub-strings it comprises encode the parameters of our reference architecture. The following data was used for each optimisation: a population of 50 individuals, a total of 4000 tests, a crossover probability of 60% and a mutation probability of 0.1%. The fitness function was a cost function (to be minimised) — $A \times P \times AAT$ for that configuration. Impossible configurations were excluded by using the approach classified in [5] as rejection of infeasible individuals.

Figure 4 shows the convergence of the GA (each point of the *P-AAT* plane is normalised to 1). As can be seen, the points are initially scattered over the range of solutions (initial population) and with the evolution of the species they get closer and closer towards the optimal solution, i.e. the one that is closest to the origin. The average number of simulated configurations is about 4000; therefore the entire optimization procedure requires CPU times less than two hours. Figure 5 shows the normalised trends of online performance (average of the results obtained in all the evaluations), offline performance (average of the best results),

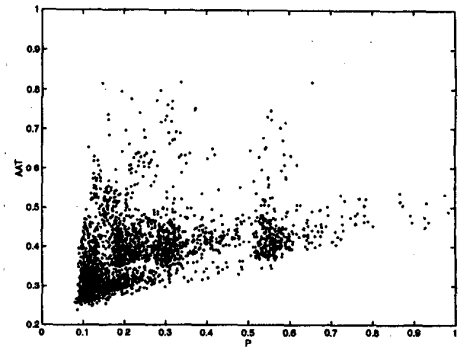

Figure 4: Evaluation results (*P-AAT* plane).

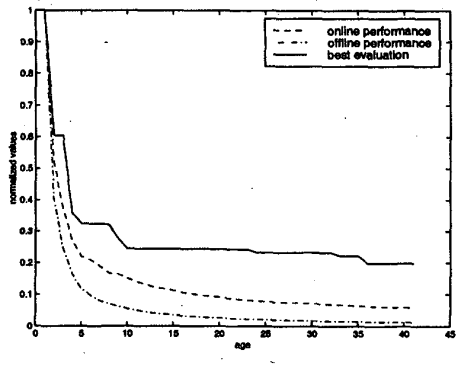

Figure 5: Online/offline performance and best result at different age.

and the best result. The curves give an idea of the convergence: a configuration that is very close to the optimal one is reached as early as the 10th period; afterwards, in fact, the improvements obtained are of less than 5%.

In general the problem of optimising several objectives at the same time does not have a single, perfect solution but a set of alternative and equally efficient solutions known as a Pareto-optimal set [8]. A more formal definition of a Pareto-optimal set is as follows: let us consider the minimisation of a function of n variables $\mathbf{f}(f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ in the universe $\mathcal{U}$. A decision vector $\mathbf{x} \in \mathcal{U}$ is said to be Pareto-optimal if and only if there is no $\mathbf{y} \in \mathcal{U}$ for which $b = \mathbf{f}(f_1(\mathbf{y}), \ldots, f_n(\mathbf{y}))$ dominates $a = \mathbf{f}(f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$, i.e., there is no $\mathbf{y} \in \mathcal{U}$ such that:

$$\forall\, i \in \{1, \ldots, n\},\ a_i \leq b_i \ \wedge\ \exists\, i \in \{1, \ldots, n\} \mid a_i < b_i$$

For each application the search of the optimal configuration is made up of the following steps: (1) definition of the cost function to be optimised, (2) GAs applications and extraction of the Pareto-optimal set (P), (3) choice of a configuration from P. By configuration we mean the configuration of the caches (in the form $S/BS/A/RP$) and the size of the bus between the first and second levels in the hierarchy (obviously the replacement policy is independent if the

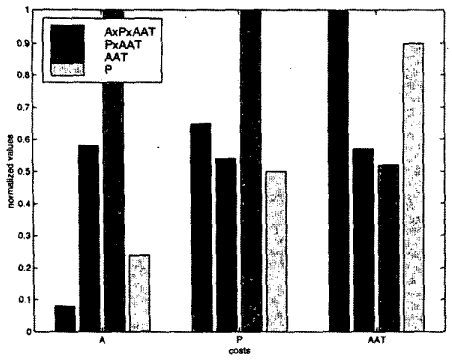

Figure 6: Optimization results summary.

associativity is 1).

5. CASE STUDY

Table 1 shows the configurations that minimise the cost $A \times P \times AAT$ for various applications. From analysis of the data it emerges that the optimal configuration greatly depends on the benchmark. More specifically, for all the benchmarks in the suite the average cache size is a first-level cache of 512 bytes for instructions, 256 bytes for data and 4K bytes for the single second-level cache. Such low values can be accounted for by the way the cost function was defined. As the size of the caches grows, the increase in area is higher than any reduction in the amount of power consumed per access and the average access time obtained with the fewer accesses to the higher levels in the hierarchy. Excluding the factor A from the cost function or adding a constraint on the area, the average cache sizes increase. The minimisation $P \times AAT$ gives an average instructions cache of 1KB, a data cache of 2KB and a second-level cache of 64KB. The average configuration needed to maximise performance is an instructions cache of 2KB, a data cache of 512 bytes and a second-level cache of 64K bytes. If the objective is to minimise power consumption, taking the workload offered by the SPEC92 suite as a reference, one needs an instructions cache of 512 bytes, a data cache of 1K bytes and a second-level cache of 16K bytes. Figure 6 summarises the results obtained for each type of optimisation performed, giving the normalised values for area, power and average access time averaged out over all the benchmarks.

Because evolutionary algorithms require scalar fitness information to work on, a scalarization of the objective vectors is always necessary. For this reason objectives are often artificially combined into a scalar function. Other optimisation techniques were also applied, such as the *compromise programming* [21] that aim to minimise the distance, not in geometric sense but in a preferential one, between a certain point and the actual achievement for each of several objectives under consideration. This point is an ideal point (normally infeasible) which corresponds to the optimum value of each objective. The solutions we found using this approach don't dominate the one we obtained using the previous cost function.

6. CONCLUSIONS

In this paper we have proposed using genetic algorithms to explore the range of configurations for a parameterisable system. The approach is an optimal one as the range of possible configurations is so great that an exhaustive search would be unfeasible. Using the genetic approach, it is possible to conduct an intelligent search for the configuration that minimises a given cost function. The approach is a completely general one. In this paper it has been applied to the search for the optimal configuration of a memory hierarchy that minimises a cost function depending on three variables area, power consumption, and average access time. Application of the method to a complex system in which there are several free parameters does not complicate matters: the only thing that needs defining is the cost function to be minimised and the coding of the genome.

From the results obtained it was observed that the optimal configuration depends heavily on the benchmark used. For this reason, the configuration of an embedded system, the functions of which are implemented by the software, can be chosen according to the code executed.

Future developments will concern two interdependent points. The first is application of the methodology to complex systems (System-On-Chip) with a large number of free parameters and consequently a vast range of possible configurations. The second is the addition of a heuristic to make the genetic algorithm more intelligent and exploit knowledge of the system being examined.

7. REFERENCES

[1] CPU info center. `http://www.eecs.berkeley.edu/CIC/`.

[2] Trace database, Parallel Architecture Research Laboratory, New Mexico State University. `http://tracebase.nmsu.edu/`.

[3] TMS320C6211 cache analysis. Texas Instruments, Sept. 1998.

[4] G. Albera and R. I. Bahar. Power/performance advantages of victim buffer in high-performance processor. In *IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, Como, Italy, Mar. 1999.

[5] C. A. C. Coello. Treating constraints as objectives for single-objective evolutionary optimization. Technical report, Laboratorio Nacional de Informatica Avanzada, Rebsamen 80, Xalapa, Veracruz 91090, Mexico, 2000.

[6] J. Edler and M. D. Hill. Dinero IV, release 7. `http://www.cs.wisc.edu/~markhill/DineroIV`, 6 Feb. 1998.

[7] M. Eisenring, E. Zitzler, and L. Thiele. Conflicting criteria in embedded system design. *IEEE Design and Test of Computers*, 17(2):51–59, Apr.–June 2000.

[8] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[9] S. Fu and M. J. Flynn. Area and performance analysis of processor configurations with scaling of technology. Technical Report CSL-TR-94-605, Computer Systems Laboratory, Stanford University, Stanford, CA 94305-4055, Mar. 1994.

benchmark	$A \times P \times AAT$	IL1	DL1	UL2	$B_{L1 \leftrightarrow L2}$
compress	0.09	1K/8/1/l	128/8/1/l	2K/8/1/l	64
eqntott	0.16	512/16/1/f	512/16/1/r	8K/16/4/l	128
espresso	0.04	512/16/1/r	128/8/1/f	4K/16/4/l	128
gcc	1	1K/16/1/r	1K/8/1/l	32K/16/4/l	128
li	0.23	4K/16/1/l	1K/16/1/l	16K/16/4/l	128
spice2g6	0.46	2K/32/1/f	4K/16/1/l	64K/32/2/l	128
doduc	0.47	8K/16/1/r	1K/8/1/l	16K/16/4/l	128
mdljdp2	0.09	512/8/1/l	256/8/1/l	1K/8/2/l	64
wave5	0.02	1K/32/1/f	256/8/1/l	2K/32/2/l	128
tomcatv	0.23	512/8/1/l	256/8/2/f	1K/16/4/l	32
ora	0.02	2K/16/1/l	512/8/1/l	4K/16/4/l	128
alvinn	0.12	4K/32/1/l	2K/8/1/l	8K/32/8/l	128
ear	0.12	512/16/1/l	512/8/1/f	8K/16/4/l	128
mdljsp2	0.12	1K/16/1/r	512/16/1/l	16K/16/4/l	128
swm256	0.02	512/16/1/l	128/8/1/l	1K/16/4/l	128
s2cor	0.23	1K/16/2/r	128/8/1/l	2K/16/4/l	64
hydro2d	0.2	256/16/1/r	128/8/1/l	1K/16/2/l	64
nasa7	0.02	512/16/1/l	128/8/1/l	1K/16/2/l	128
fpppp	0.66	512/16/1/f	2K/8/1/l	16K/16/4/l	128

Table 1: Best configurations that minimize the cost $A \times P \times AAT$.

[10] J. D. Gee, M. D. Hill, D. N. Pnevmatikatos, and A. J. Smith. Cache performance of the SPEC92 benchmark suite. *IEEE Micro*, 13(4):17–27, Aug. 1993.

[11] T. Givargis and F. Vahid. Interface exploration for reduced power in core-based systems. In *International Symposium on System Synthesis*, Dec. 1998.

[12] T. D. Givargis, J. Henkel, and F. Vahid. Interface and cache power exploration for core-based embedded system design. In *International Conference on Computer-Aided Design (ICCAD)*, pages 270–273, Nov. 1999.

[13] T. D. Givargis and F. Vahid. Parametrized system design. In *8th International Workshop on Hardware/Software Codesign*, 2000.

[14] T. D. Givargis, F. Vahid, and J. Henkel. A hybrid approach for core-based system-level power modeling. In *Asia and South Pacific Design Automation Conference*, 2000.

[15] J. J. Grefenstette. *A User's Guide to GENESIS*, Oct. 1990.

[16] J. L. Hennessy and D. A. Patterson. *Computer Architecture A Quantitative Approach*, pages 372–476. Morgan Kaufmann, second edition, 1996.

[17] D. Kröning and S. M. Müller. The impact of write-back on the cache performance. In *Proc. 18th IASTED International Conference on Applied Informatics, Innsbruck (AI'2000)*, pages 213–217. ACTA Press, 2000.

[18] E. Musoll, T. Lang, and J. Cortadella. Working-zone encoding for reducing the energy in microprocessor address buses. *IEEE Transactions on Very Large Scale Integration (VLSI)*, 6(4), Dec. 1998.

[19] O. Okuzawa and M. J. Flynn. A performance/area workbench for cache memory design. Technical Report CSL-TR-94-635, Computer Systems Laboratory, Stanford University, Stanford, CA 94305-4055, Aug. 1994.

[20] G. Reinman and N. Jouppi. An integrated cache timing and power model. Technical report, COMPAQ Western Research Lab, Palo Alto, 1999.

[21] C. Romero, M. Tamiz, and D. Jones. Goal programming, compromise programming and reference point method formulations: linkages and utility interpretations. *Journal of the Operational Research Society*, (49):986–991, 1998.

[22] Synopsys, Inc. CoWare, Inc. Frontier Design, Inc. *SystemC 0.91 User's Guide*.

[23] F. Vahid and T. Givargis. The case for a configure-and-execute paradigm. In *International Workshop on Hardware/Software Codesign (CODES)*, pages 59–63, May 1999.