Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2011, Article ID 620578, 12 pages
doi:10.1155/2011/620578

Research Article

A Combined Optimization Method for Tuning Two-Level Memory Hierarchy Considering Energy Consumption

Abel Guilhermino Silva-Filho and Filipe Rolim Cordeiro

Informatics Center (CIn), Federal University of Pernambuco (UFPE), 50740-540 Recife, PE, Brazil

Correspondence should be addressed to Abel Guilhermino Silva-Filho, agsf@cin.ufpe.br

Received 25 May 2010; Revised 24 August 2010; Accepted 21 September 2010

Academic Editor: Xiaorui Wang

Tuning cache hierarchies in platforms for embedded systems can significantly reduce energy consumption. In this paper we combined two optimization methods for tuning both instruction and data cache configurations in a two-level memory hierarchy, where both levels have separate instruction and data caches. This kind of hierarchy allows us to evaluate instruction and data caches branches separately, although previous approaches have applied the same method for both branches of the hierarchy. This work evaluates several methods intended for two-level hierarchies, and the results showed that when we combine different methods for each branch of the hierarchy, results can be improved. Experiments based on simulations were performed for 12 applications from the Mibench suite benchmark and the combined method achieved better efficiency in 60% of the evaluated cases compared with existing heuristics. The proposed solution is only 11% less economic in terms of energy consumption than optimal values and required, on average, 42 simulations to conclude optimization mechanism, representing only 9% of the design space.

1. Introduction

The memory subsystem has been demonstrated to be the energy bottleneck: several researchers [1, 2] have demonstrated that the memory subsystem now accounts for 50%–70% of the total power budget of the system [3]. Thus, many efforts have been made to develop optimization mechanisms in order to reduce energy consumption without degrading the performance.

Recent technologies have enabled the tuning of cache memory parameters on the basis of core-based processors for the needs of an application [4–7]. A suitable choice for a given combination of cache configuration for a specific application can lead to energy saving without compromising time performance and can be configured during system reset or even during runtime [8, 9].

However, an exhaustive approach can take a very long time to reach convergence, due to the large number of possible combinations in the design space. For instance, in hierarchies with only one cache level, it is necessary to explore the design space of dozens of configurations when the total cache size, line size, and associativity parameters variations are considered. In these cases, an exhaustive approach is feasible, because cost and simulation time are low. In hierarchies that include a second level of cache, where both levels have separate instruction and data caches (the focus of this work), some hundreds of configurations would need to be explored when the same parameters were considered. Moreover, for hierarchies that have a unified second-level cache (instructions and data), this number can total thousands of configurations, owing to the interdependence between data and instructions at the second level [10]. The exhaustive approach stops being a good option when exploration space increases exponentially, due to addition of new parameters, such as processor, transistor technology, or when simulation time is too long. For those cases, an intelligent mechanism is more appropriated to find good solutions to the hierarchy. Even simulation time being cheap in some cases, when exploration involves thousands of simulations, other options such as heuristics can be more interesting. Applying hybrid mechanisms may be indicated when both levels have separated instruction and data caches

because the optimization can be applied independently for distinct branches. In this case, the configuration of one cache hierarchy does not greatly affect the other cache hierarchy.

In addition, in some cases, a given optimization mechanism can be good for instruction cache exploration, but this is not true for data cache exploration. In this sense, hybrid efficient optimization mechanisms intended for each specialty, associated with an environment capable of analyzing the behavior of the architecture, can enhance and speed up the memory hierarchy design.

In this paper, we combine two optimization mechanisms for a highly configurable two-level cache hierarchy, where both levels have separate instruction and data caches. We combine previous methods intended for instruction and data caches separately in order to improve the results in terms of energy consumption and performance.

In the next section, we discuss the background details on architecture exploration based on some recent related work. In Section 3 we introduce some considerations on preliminary studies performed before proposing the hybrid optimization mechanism. Section 4 presents the combined optimization method specialized for instruction and data caches. Section 5 presents the results of the proposed approach for 12 distinct applications from MiBench [11] suite, and finally, in Section 6 we discuss the conclusions and future directions.

2. Background

Typically, some design objectives such as area, performance, and power compete with one another; that is, improving one often leads to worsening another. For example, if we increase an implementation's performance, the implementation's energy consumption may suffer.

Thus, usually, parameterized SoC architectures should be optimally tuned to meet multiple design objectives in a large class or subset class of applications. Efficient design space exploration mechanisms are therefore necessary to reduce costs when searching for optimal configurations in the large exploration space.

Typical parameterized system-on-chip architectures are composed of a processor core, one or more cache memory units, on-chip bus hierarchy, on-chip memory, and other peripheral cores with application-specific functionality. Each of these components can receive one or more parameters. The collection of all possible configurations represents the *configuration space*. For two-level caches in the memory hierarchy, the amount of parameters may considerably increase the configuration space.

Gordon-Ross et al. [9] have observed poor results when applying the heuristic to two-level caches (with separated instruction and data caches for both levels). Gordon-Ross et al. [9] have extended Zhang's heuristic [12] (intended for one-level cache) and proposed the TCaT heuristic. The heuristic interlaces the exploration of the two cache levels and searches the various cache parameters in a specific order, based on their impact on energy. Initially the cache parameters are set to be minimal and are explored toward maximal values. The heuristic begins by exploring the cache size of the first level and then that of the second level. An exploration of the cache line size of the first and then second levels is performed in sequence. Finally, the heuristic explores both (first and then second) levels of associativity. While the exploration of one parameter is good for hierarchy in terms of energy, that parameter keeps varying and the others remain fixed. Once the exploration of a parameter stops providing good solutions or the maximum parameter value is reached, another parameter is explored, fixing the others, until all parameters have been tuned following the sequence described previously. This procedure is done first for instruction caches and afterwards repeated for data caches. The use of the TCaT heuristic allows energy savings of 53% when compared with Zhang's heuristic.

Silva-Filho et al. [13] presented a heuristic for two-level caches, known as TECH. Preliminary results have demonstrated that the order in which cache parameters are explored affects the number of states that should be visited to find the best configuration for a given application. The heuristic is focused on energy consumption. Basically, the TECH heuristic is similar to TCaT, but it adopts a reverse exploration order and a different exploration environment based on the Architecture Description Language known as ArchC. TECH explores the second level of the cache hierarchy before the first level. In TECH heuristic, all cache parameters are set initially to be minimal, with the exception of second level cache size, that is set to be maximal. The exploration of parameters follows the same logic of TCaT heuristic, but the sequence of exploration of parameters starts tuning cache size of second level, followed by first level. After that, the exploration is done for line size of second level and then first level. At last, the heuristic is concluded exploring associativity of second level, followed by the first level. Through simulations it was observed that to define the cache size of the second level, initially, we need to limit increases in the first cache level. Preliminary experiments show an approximately 4-fold reduction in energy consumption when compared with TCaT heuristics.

Silva-Filho et al. [14] proposed improvements to both TCaT and TECH approaches. They presented a method for two-level caches known as TECH-CYCLES. Unlike previous approaches, this heuristic considers two different objectives: minimize energy consumption and improve the time processing of the application. Basically, TECH-CYCLES heuristic is similar to TECH heuristic, but besides considering the impact of energy consumption to determine if a solution is better than other, it also considers the time processing of an application. In practical terms, the exploration of a parameter in TECH-CYCLES continues while values of energy consumption and cycles of an application are better than last explorated parameters. The results from this approach have shown an average reduction of about 41% in energy consumption for instruction caches and an improvement of about 25% in time processing when compared with the TCaT heuristic. However, the method applied for data caches did not achieve good results in terms of time processing and energy consumption.

Subsequently, Silva-Filho et al. [15] proposed the TEMGA optimization mechanism, also focused for two-level memory hierarchy (with separated instruction and data caches for both levels). This approach is based on a simple genetic algorithm (GA) and intended for two-level data caches. The mapping of a solution, also called chromosome, is done through the definition of cache parameters to each solution. So, each solution is represented as a set of six parameters (cache size, line size, and associativity for first and second levels), which are associated with a value of energy and cycles necessary to run an application. Through GA execution, initially are created random solutions, which represent cache configurations, with a value of energy and cycles associated. These solutions are submitted to crossover and mutation operators. In crossover mechanism two solutions are combined to produce a new one, preserving characteristics (cache parameters) of the initial ones. This permits to create solutions combining good characteristic of existing solutions, such as cache size or associativity. The mutation operator is responsible for changing a random parameter of a solution, in order to promote diversity of solutions. In the end a selection mechanism is applied in order to select the best solutions in terms of energy consumption and cycles to run an application. The selected solutions are preserved and then submitted to crossover and mutation operators. This process is repeated until the new solutions have no significant improvement compared to the previous ones. The results of this approach show an average reduction of about 15% in energy consumption for data caches when compared with the TECH-CYCLES and TCaT heuristics. In addition, a 5-fold reduction in the number of cycles needed to execute applications from Mibench was observed [11].

As can be seen, there are a number of mechanisms with similar objectives that present better efficiency when applied only for data caches (TEMGA), others when applied only for instruction cache (TECH-CYLES), and also those that were applied for both (TCaT, TECH, and Zhang heuristics).

This paper aims to combine state-of-the-art mechanisms intended for instruction and data caches in a two-level hierarchy (with separated instruction and data caches for both levels) resulting in a combined optimization mechanism that determines the most suitable relation between energy consumption and performance applications. Comparisons between previously reported optimization mechanisms were analyzed in terms of their specialties for data and instruction caches and some results are shown in the next section.

3. Preliminary Studies

A number of studies on optimization mechanisms intended for two-level cache hierarchies considering separate instruction and data caches for both levels are described in this section. In addition, the environment used to simulate all benchmarks and some considerations on energy consumption and performance intended for two-level hierarchies are also discussed.

Table 1: Configuration space for the cache hierarchy.

Parameters	Cache level 1 (instruction and data)	Cache level 2 (instruction and data)
Cache size	2 Kb, 4 Kb, 8 Kb	16 Kb, 32 Kb, 64 Kb
Line size	16B, 32 B, 64 B	16 B, 32 B, 64 B
Associativity	1, 2, 4	1, 2, 4

3.1. Experimental Setup Environment. The environment used to simulate all applications is composed of a System-on-Chip architecture with an MIPS core processor, a one-level cache with independent instruction (IC1) and data (DM1) caches, a two-level cache with independent instruction (IC2) and data (DM2) caches, and main memory (MEM). An input voltage of 1.7 V is used, with a write-through scheme and transistor technology 6-T of 0.08 μm. Such studies can also be applied for smaller transistor technologies. However, with the aim of making comparisons with previous research studies, we adopted the same transistor technology.

Independent analyses in terms of performance and energy consumption may be done for each branch of the hierarchy (instruction cache branch or data cache branch). A given branch may be defined by a pair of caches (level-one and level-two caches). Twelve applications from the MiBench benchmark suite [16] have been considered in the experiments of the proposed approach.

Each cache configuration is compiled and simulated by using the SimpleScalar [16] tool in order to determine the numbers of misses, hits, and accesses to the memory hierarchy. Furthermore, the energy consumption of the hierarchy is determined using the eCACTI cache memory model [17]. This model evaluates two energy components: static and dynamic. In fact, considering an energy cache model without the static energy component can cause significant inaccuracies for recent transistor technologies. In accordance with the ITRS predictions, the static energy component, which was negligible in previous technologies, represents up to 50% of the energy in CMOS circuits [18], and it is rapidly increasing when compared with the dynamic component. Despite the use of the eCACTI cache memory model, other models such as CACTI (version 5.x above) [19] also can be used.

Typical commercial cache configurations for embedded applications were considered in the exploration space (see Table 1). The parameters indicated for each cache level in Table 1 limit the configuration space for instruction and data caches. Considering only variations in the cache parameters (total cache size, cache line size, and associativity) for both cache levels, the total configuration space for a given application has about 500 different configurations.

3.2. Comparisons between Memory Hierarchies. A number of aspects mentioned in this subsection can be used to justify the studies of exploration mechanisms for hierarchy with two-level caches (with separated instruction and data caches for both levels).

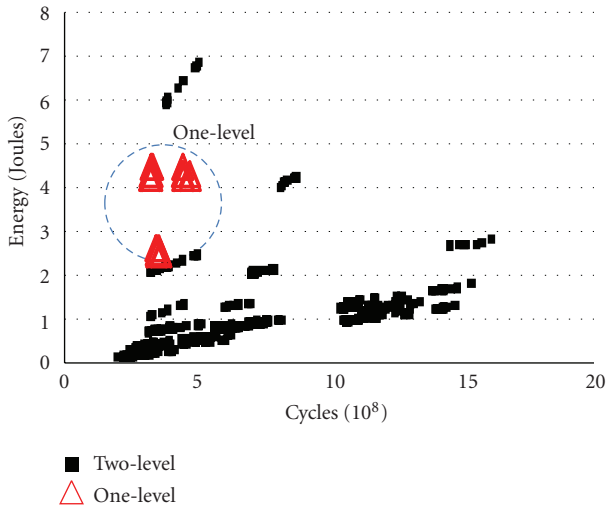

Figure 1: Comparison between one- and two-level hierarchies considering *patricia_small* and the instruction branch.

Some important aspects are disclosed when two-level memory hierarchy and one-level cache hierarchy are compared. The use of the two-level hierarchy approach (hundreds of configurations) may be an unsatisfactory solution in architecture exploration when an inadequate optimization mechanism is applied or an exhaustive approach is used to search for the best configuration [12, 20]. The small exploration space of the one-level approach (dozens of configurations) leads to an exhaustive approach or the use of simple heuristics. However, the exploration space of each approach may be located in different regions of the space (Figure 1). In this connection, even the complete small space of configurations of the one-level hierarchy approach is sometimes available; there are many configurations in the two-level hierarchy approach with energy and performance values, for instance, that are better than the one-level hierarchy approach. Clearly that could not happen for all kinds of application, but it is an aspect that encourages us to develop optimization mechanisms that speed up the search in the design space by achieving suitable hierarchies configurations.

The design space defined for single- and two-level hierarchies was exhaustively simulated according to the experimental environment defined in the previous subsection. The exhaustive approach for the two-level cache considers exactly 458 different configurations, and for a single-level cache, it considers 24 different configurations. By considering a single application from the MiBench benchmark suite [16], namely, *patricia_small*, we exhaustively compared both approaches. Figure 1 depicts the configuration space exploration for such an application.

Each point in Figure 1 represents the energy and the number of cycles estimated for a given configuration. The triangles in the dotted circle represent the complete design space for a one-level cache hierarchy. The squares in the rest of the figure represent the complete design space for two-level cache hierarchy. This shows that by using a one-level cache hierarchy we can obtain several configurations having a high performance (evaluated by the reduced number of cycles). However such high-performance points demand a high energy consumption. On the other hand, we have two-level cache configuration points with low energy consumption, but with several points with a reduced number of cycles, resulting in a rapid execution of the application.

The results presented for this application revealed that by using memory hierarchies with a two-level cache, we can improve not only the performance but also the energy consumption needed to run the application.

An additional study in order to analyze performance and energy consumption for different applications, considering two types of hierarchy with two levels, specifically with the second level unified and separate, was performed.

Some considerations are important to mention about memory hierarchy with unified second level. The advantage of this architecture is more flexible use of cache storage. However, since CPU accesses instructions and data in different pattern, caching these together may pollute the cache and degrade cache performance for instructions and/or data. Therefore, split caches for instructions and data are used, to best utilize the accessing pattern.

Recently, some commercial approaches such as Montecito from Intel's Itanium 2 Processor family [21, 22], with 90 nm process, allowed for dual core implementation, have improved the memory hierarchy and allowed it to become reasonably competitive. This approach, despite it includes an L3 cache per core, has separate instruction and data for L1 and L2 caches. Another approach, also from Intel and Itanium 9300 series [22] which is the Tukwila, with 65 nm process, has separate instruction and data for L1 and L2 caches.

Additionally, depending on the nature of the application it can result in different behaviors, impacting on performance and energy consumption of the application. Four examples are shown in the figures comparing architectures with two levels in the memory hierarchy, considering the unified and separated second level. Each point in the figure represents the energy and the number of cycles estimated for a given configuration. The diamonds represent the complete design space for a unified cache hierarchy. The squares in the rest of the figure represent the complete design space for separate cache hierarchy. Figure 2 illustrates that by using separate second-level approach we can obtain configurations with high performance and energy consumption compatible with unified approach. These solutions considering *susan_small* application for 70 nm process may be shown in dashed circle.

Figure 3 illustrates another case using *djkistra_small* application also for 70 nm technology that shows compatible results in terms of energy consumption and performance for both approaches, being the separate approach slightly better than the unified approach, as can be shown in the figure.

Figure 4 illustrates another case using *sha_small* application also for 70 nm technology that shows better results in terms of energy consumption and performance for separate approach when compared with unified approach, as can be shown in dashed circle.

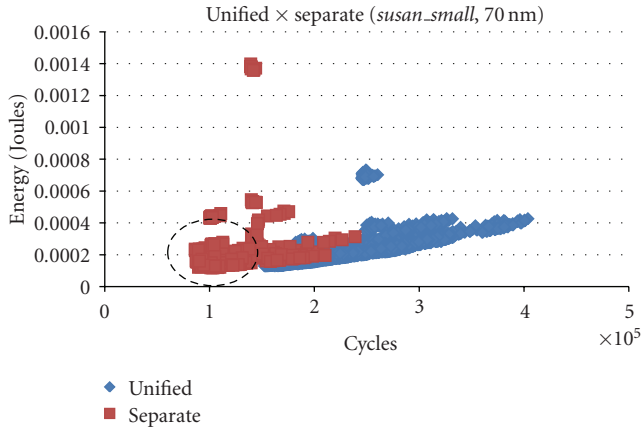

Figure 2: Comparison between two-level hierarchies with separate and unified second level considering *susan_small* and 70 nm.

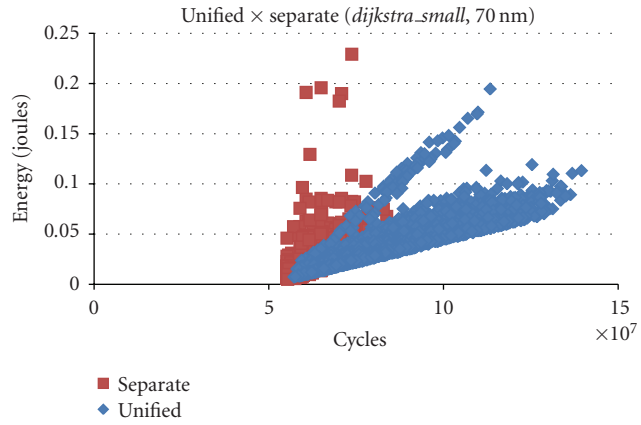

Figure 3: Comparison between two-level hierarchies with separate and unified second level considering *djkistra_small* and 70 nm.

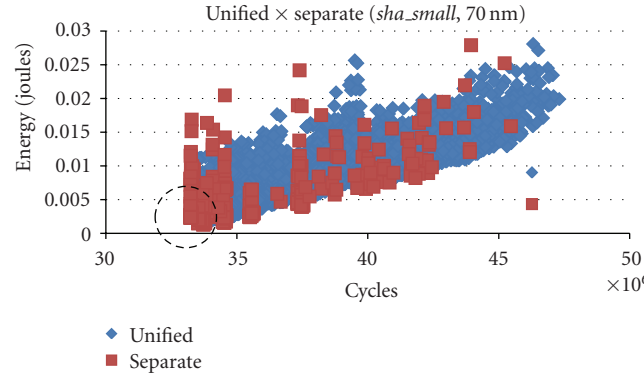

Figure 4: Comparison between two-level hierarchies with separate and unified second level considering *djkistra_small* and 70 nm.

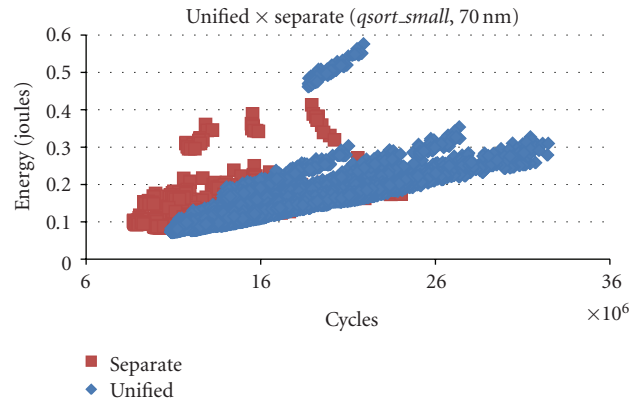

Figure 5: Comparison between two-level hierarchies with separate and unified second level considering *qsort_small* and 70 nm.

Figure 5 illustrates results for *qsort_small* application that is being used to compare both two-level hierarchies approaches. Results for this case show that unified approach had better results in terms of energy consumption when compared with separate approach; however in terms of performance separate approach presents better results than unified approach as can be shown in the figure.

Furthermore, there are other aspects such as transistor technology reduction that can directly influence the energy and performance application. Unfortunately, we did not have much time to simulate many applications considering this aspect, but Figure 6 illustrates results for *crc32_small* application considering 180 nm and 70 nm transistor technologies. Results show that separate approach presents better Pareto optimal when compared with unified approach when transistor technology is reduced.

As can be shown in previous analysis, the separate approach that considers a two-level hierarchy with data and instructions separate for both levels presents good results when compared with unified approach in the most cases presented. Although only four cases are shown, results show that there are still viable studies considering hierarchies with separate approach.

3.3. Comparison between Mechanisms. On the other hand, different analyses for different branches of the hierarchy can provide an improvement in the results in terms of performance and energy consumption. Our studies succeeded in showing that a given optimization mechanism may not be particularly appropriated for both branches of the hierarchy.

The following optimization mechanisms were considered in our analyses: TECH [13], TCaT [9], TECH-CYCLES [14], and TEMGA [15]. We exhaustively simulated the design space defined for two-level cache hierarchies according to the experimental environment defined in the previous subsection in order to better evaluate the results obtained.

Twelve applications from the Mibench benchmark suite were considered. To evaluate the impact for different applications in terms of two objectives (energy and performance), we adopted the cost function represented by $F = Energy \times Cycles$. Low energy consumption associated with a low number of cycles is two objectives that one wishes to attain. Thus, minimal values for the cost function (F) are considered cache configurations that produce suitable solutions in the design space.

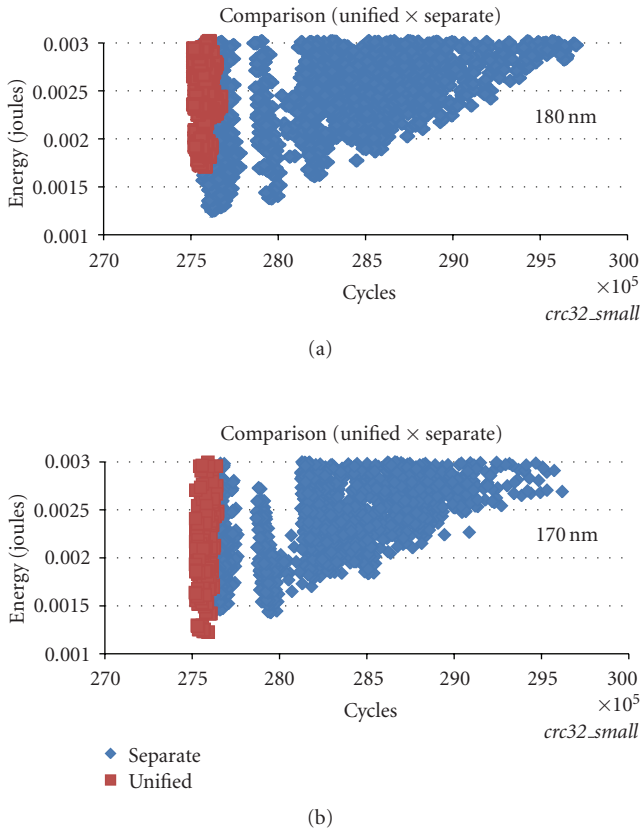

Figure 6: 180 nm and 70 nm transistor technologies comparison for separate and unified second level considering *crc32_small*.

Applications from the Mibench benchmark suite were analyzed in terms of cost function for different heuristics considering simulation in different branches. For each branch in the hierarchy (instruction branch and data branch) all the previously mentioned heuristics were applied in order to obtain the best heuristic for a given branch.

By means of this study it was possible to choose a suitable heuristic for the instruction and data branches. As shown in Table 2, some mechanisms achieved the same results. For instance, the *susan_small* application achieved the same result (hierarchy configuration) for the instruction branch by applying both the TECH and TEMGA mechanisms. The *susan_large* application achieved the same result (hierarchy configuration) for the data branch when TECH, TCAT, and TEMGA mechanisms were applied. However, we consider the largest percentile average value as the best mechanism for each branch.

Table 3 summarizes this analysis, considering the number of best occurrences. The TECH-CYCLES was the best mechanism achieving about 50%, on average, the best results for all evaluated applications in the instruction branch. A similar analysis was done for the data branch, and TEMGA was shown to be the best mechanism achieving the best results for approximately 83% of the applications analyzed.

Table 2: Comparison between different heuristics for two-level hierarchy with separated instruction and data caches for both levels.

Application (from Mibench suite)	Best mechanism (Instruction branch)	Best mechanism (Data branch)
basicmath_small	TECH-CYCLES	TEMGA
basicmath_large	TECH-CYCLES	TECH
bitcount_small	TEMGA	TECH/TEMGA
bitcount_large	TEMGA	TECH/TEMGA
dijkstra_small	TEMGA	TEMGA
dijkstra_large	TECH-CYCLES	TECH-CYCLES
patricia_small	TECH-CYCLES	TEMGA
patricia_large	TECH-CYCLES	TEMGA
qsort_small	TEMGA	TEMGA
qsort_large	TECH-CYCLES	TEMGA
susan_small	TEMGA/TECH	TCAT/TEMGA
susan_large	TECH	TECH/TCAT/ TEMGA

Table 3: Percentile average value evaluation for data and instruction branches and different optimization mechanisms.

Mechanism	Instruction branch	Data branch
TECH	17%	42%
TCAT	0%	17%
TECH-CYCLES	**50**%	17%
TEMGA	42%	**83**%

4. Proposed Mechanism Evaluation

We hereby propose a combined optimization mechanism, named TC-HyOM, intended for two-level caches with separated instruction and data caches for both levels and based on the analysis described in the previous section. The proposed mechanism is a software-based solution, which is intended to find the best configurations to an application through combined simulation of different techniques.

The proposed method allows two-level-cache exploration, both levels having separate instruction and data caches; however, instruction and data caches are evaluated separately, each one with a different optimization mechanism intended for each hierarchy, that is, TECH-CYCLES for the instruction branch and TEMGA for the data branch, respectively.

4.1. Instruction Branch Analysis (TECH-CYCLES Heuristic). Based on preliminary studies described in the previous section, the TECH-CYCLES heuristic was chosen to determine the configuration of the instruction cache for both first and second levels of the hierarchy of the TC-HyOM-proposed mechanism.

A number of optimization mechanisms (TECH-CYCLES [14], TECH [13], TCaT [9], and TEMGA [15]) were applied to the selected applications by using the previously mentioned experimental environment, and the impact in

terms of energy consumption was also evaluated. These mechanisms use the same exploration environment for both branches (data and instruction).

For comparison purposes, we chose a base cache hierarchy configuration defined by the 8-kByte, 4-way set associative level one cache with a 64-Byte line size, and a 64-kByte, 4-way set associative level two cache with a 64-Byte line size—a fairly common configuration.

Table 4 shows the instruction branch energy consumption normalized to base cache and 12 applications from Mibench using (1). The results for the instruction branch indicated that, on average, the TECH-CYCLES heuristic consumes less energy compared with the other heuristics analyzed:

$$\text{Energy}_{\text{Normalized}} = \frac{\text{Energy}_{\text{Heuristics}}}{\text{Energy}_{\text{Base\_Cache}}}. \quad (1)$$

For all analyzed applications (instruction branch), we obtained a reduction in energy consumption compared with the base cache. On average, an energy reduction of around 47% is achieved when TECH-CYCLES is compared with base cache. In some cases it exceeded 80% (*bitcount_small, bitcount_large, susan_small,* and *susan_large*).

We also evaluated the impact in terms of performance. Thus, an analysis in terms of a cost function was done. We adopted the cost function represented by $F = Energy \times Cycles$. We consider that minimal values for the cost function produce cache configurations close to optimal-Pareto. The optimal-Pareto region is defined by configurations that produce the best tradeoff between energy and number of cycles. Thus, the optimal-Pareto solution set represents the best solutions of exploration space. The configuration with the lowest cost (*Energy* $\times$ *Cycles)* is identified as the best tradeoff relation between energy and number of cycles. The results show, on average, a reduction in terms of cost function of around 50% when TECH-CYCLES is compared with the base cache.

In order to show the effectiveness of the instruction branch tuning heuristics in reducing energy, we compared the results of TECH, TCAT, TECH-CYCLES, and TEMGA heuristics with optimal results. The optimal results were found by exhaustive exploration and are obtained by selecting the solution with the lowest cost (*Energy* $\times$ *Cycles)* in exploration space, for each application. Exhaustive exploration is not necessary for running the analyzed heuristics. It was done just as a way to measure the quality of solutions found by each heuristic.

Each column in Figure 8 represents the best configuration for the selected instruction branch in a given optimization mechanism in terms of energy consumption. The last column represents the lowest energy consumption value for a given application (indicated as optimal in Figure 8). Energy consumption for each configuration is normalized to energy consumption of the base cache for that application.

In order to normalize the data for comparison purposes, we chose the same base cache hierarchy configuration defined previously. Five out of twelve applications analyzed (*basicmath_small, basicmath_large, bitcount_large, susan_large,* and *patricia_small*) achieved the lowest energy consumption results when the TECH-CYCLES heuristic was applied for the instruction branch (Figure 8).

Table 4: Energy consumption for instruction branch normalized to base cache and compared with other heuristics.

Application (from Mibench suite)	TECH	TCAT	TECH-CYCLES	TEMGA
basicmath_small	13.7250	13.7250	0.8702	0.8702
basicmath_large	2.2498	4.1856	0.8523	1.1803
bitcount_small	0.2018	0.2277	0.1881	0.1871
bitcount_large	0.1668	0.1929	0.1659	0.1658
dijkstra_small	0.5852	0.6850	0.5506	0.4790
dijkstra_large	0.4103	0.5233	0.3616	0.4371
patricia_small	7.6043	7.7764	0.8371	0.8829
patricia_large	7.4594	7.6728	0.8296	1.0779
qsort_small	0.9067	0.9149	0.7276	0.6084
qsort_large	1.1287	0.8627	0.5796	0.8514
susan_small	0.1774	0.2086	0.1949	0.1774
susan_large	0.1654	0.1920	0.1667	0.1666
Average	2.8984	3.0972	0.5270	0.5905

In addition, we evaluated the impact of the number of simulations necessary to conclude the execution for each heuristic in the instruction branch. We observed an average of 9, 11, 10, and 54 simulations for TCaT, TECH, TECH-CYCLES, and TEMGA heuristics, respectively, considering twelve applications from the Mibench benchmark. Figure 7 shows these results, and we can see that TCaT, TECH, and TECH-CYCLES heuristics have similar numbers and represent approximately 0.3% of the search space, which contains exactly 458 configurations. On the other hand, the TEMGA heuristic, on average, concludes the optimization mechanism for the instruction branch at 54 simulations and represents approximately 12% of the same search space.

A total of about 25 hours was spent in simulating all applications with the TEMGA heuristic and about 4.4 hours in concluding the other heuristics (TCaT, TECH, and TECH-CYCLES) for instruction branch, demonstrating that it is an attractive solution when compared to the 210 hours that would be necessary to simulate an exhaustive approach. We applied all heuristics in a Pentium IV machine with 256 MB RAM memory.

Similar analysis was done for data branch and will be shown in the next section.

These results reveal TECH-CYCLES as a suitable approach for the instruction branch that has, on average, a low energy consumption, a good tradeoff relation when performance is evaluated and included in the analysis, and, finally, a low number of cycles needed to conclude the optimization mechanism.

4.2. Data Branch Analysis (TEMGA Heuristic). The TEMGA heuristic was chosen to be applied on the first and second levels of the data branch of the TC-HyOM-proposed

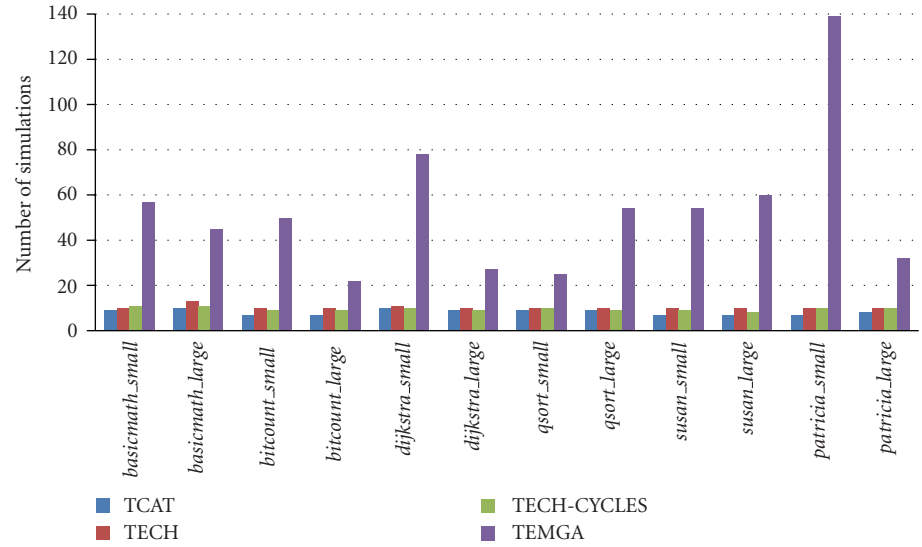

Figure 7: Number of simulations for instruction branch and for different optimization mechanisms.

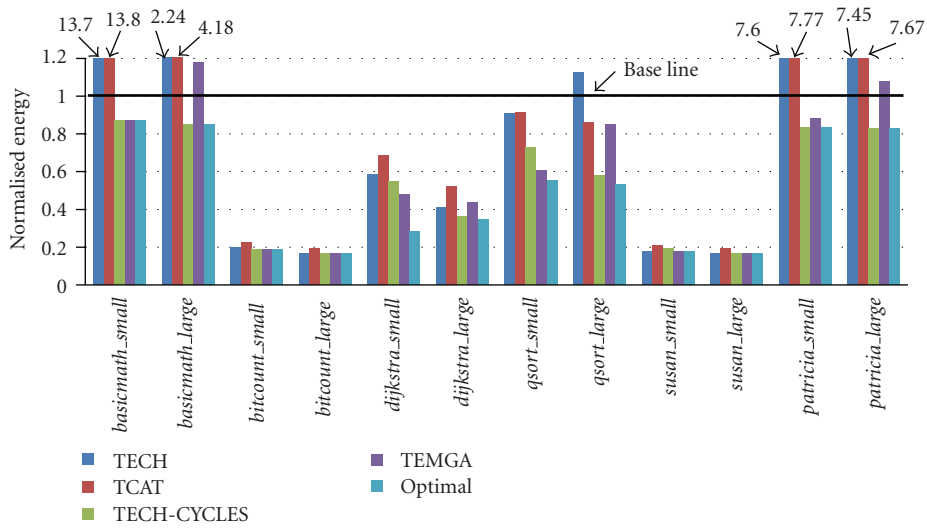

Figure 8: Energy consumption for instruction branch normalized to the base cache configuration for different optimization mechanisms (TECH, TCAT, TECH-CYCLES, and TEMGA) and optimal values.

mechanism, based on comparative studies mentioned in the previous section. TEMGA is based on Genetic Algorithms (GAs), which is an adaptive approach based on the evolution of a group of individuals undergoing genetic changes and does not require a detailed knowledge of the problem.

A similar analysis was done for the data branches, also using the same optimization mechanisms (TECH-CYCLES [14], TECH [13], TCaT [9], and TEMGA [15]) and environment setup to evaluate the impact in terms of energy consumption of these mechanisms in the data branch.

In addition, for comparison purposes, we chose the same base cache hierarchy configuration defined previously for the instruction branch analysis. Table 5 shows the data branch energy consumption normalized to the base cache and 12 applications from Mibench, also according to (1). The results for the data branch indicated that, on average, TEMGA heuristic consumes less energy consumption compared with the other heuristics analyzed.

When the data branch is evaluated in terms of cost function ($F = Energy \times Cycles$), the results show that, on average, TEMGA achieves a reduction of around 44% when compared with the base cache, considering all optimization mechanisms and 12 applications from Mibench. As with the data branch analysis, the configuration with the lowest cost ($Energy \times Cycles$) is identified as the best tradeoff relation between energy and number of cycles. The TEMGA heuristic achieves good results in terms of energy consumption for the majority of applications except in three cases (basicmath_large, djkistra_large, and qsort_large).

Table 5: Energy consumption for data branch normalized to base cache and compared with other heuristics.

Application (from Mibench suite)	TECH	TCAT	TECH-CYCLES	TEMGA
basicmath_small	0.5865	0.5473	0.5987	0.5460
basicmath_large	0.4666	0.5321	0.4725	0.5321
bitcount_small	0.1828	0.2109	0.1949	0.1828
bitcount_large	0.1656	0.1920	0.1665	0.1656
djkstra_small	2.9523	2.9866	0.7973	0.7912
djkstra_large	3.3156	3.2969	0.8042	1.0442
patricia_small	0.9077	0.8701	0.9266	0.6983
patricia_large	0.9554	0.8898	1.0592	0.8057
qsort_small	0.6738	0.6500	0.9488	0.5895
qsort_large	0.5983	0.6622	0.9319	0.6703
susan_small	0.4774	0.4122	0.5503	0.4122
susan_large	0.4427	0.4427	0.5038	0.4427
Average	0.9771	0.9744	0.6629	0.5734

The memory hierarchies that achieved the lowest energy consumption values for the data branch, known as optimal value, were compared in terms of energy consumption with all the above mentioned heuristics (Figure 9). Energy consumption for each configuration is normalized to energy consumption of the base cache for that application. The base cache is the same as that applied to the instruction branch.

We can see that the TEMGA heuristic achieves the best results in terms of energy consumption for two applications (*susan_small* and *patricia_small*). For some applications (*susan_large, qsort_small, bitcount_large,* and *bitcount_small*), despite not achieving optimal values, the energy consumption values are very close to optimal ones.

As with the data branch, the impact of the number of simulations necessary to conclude the execution for each heuristic was evaluated. We observed an average of 9, 12, 10, and 33 simulations for TCaT, TECH, TECH-CYCLES, and TEMGA heuristics, respectively, and twelve applications from the Mibench benchmark. Figure 10 shows these results and we can see that TCaT, TECH, and TECH-CYCLES heuristics produce values similar to those of the instruction branch in terms of the number of simulations and represent approximately 0.3% of the search space.

However, for the TEMGA heuristic, this average for data branch falls to 33 simulations when compared with the same mechanism applied to the instruction branch (54 simulations) and represents approximately 7.2% for the same search space.

On average, a total of about 14 hours was spent on simulating all applications with the TEMGA heuristic and about 3 hours on concluding the other heuristics (TCaT, TECH, and TECH-CYCLES) for the data branch. As with the data branch, we applied all heuristics in a Pentium IV machine with 256 MB RAM memory.

Although the TECH-CYCLES, TECH, and TCaT heuristics are faster in executing the heuristic, TEMGA found individuals who spent, on average, 40%, 40%, and 13% less energy when compared with the results obtained by the TECH, TCaT, and TECH-CYCLES, respectively.

Based on these results, the TEMGA optimization mechanism, when used for the data branch in a two-level memory hierarchy with separated instruction and data caches for both levels, has been shown to be an interesting approach that finds individuals with low energy consumption and shows a good tradeoff relation when performance is evaluated. In addition, despite the large number of simulations needed to conclude the optimization mechanism, this kind of approach does not require a detailed knowledge of the problem, as it is based on the natural selection proposed by Charles Darwin, in which the fittest individuals in a given population have the best chance of reproducing and surviving.

In the next section, we describe some results for the proposed approach (TC-HyOM) that associate TECH-CYCLES and TEMGA as optimization mechanisms for instruction and data branches, respectively.

5. Results of the TC-HyOM

The proposed optimization mechanism Two-level Cache Hybrid Optimization Mechanism (TC-HyOM) intended for two-level cache hierarchies with separated instruction and data caches for both levels was validated by applying twelve applications from the Mibench suite in the experimental environment mentioned in Section 3.1.

In this work, the instruction and data branches were analyzed separately and the optimizations mechanisms were observed by applying each mechanism for both branches individually. For analysis of the TC-HyOM optimization mechanism, the total energy consumption of the memory hierarchy needs to be obtained. Table 6 shows the total energy consumption normalized to the base cache for the TC-HyOM, TECH, TCaT, TECH-CYCLES, and TEMGA optimization mechanisms. The second to last and last lines of the table include the average and summation of the total energy consumption, respectively, also normalized to the base cache.

In this case, each optimization mechanism (TECH, TCaT, TECH-CYCLES, and TEMGA) was applied for both branches (instruction and data) and added to obtain the total energy consumption of the memory hierarchy. In addition, the last column (the hybrid approach TC-HyOM) was evaluated with the proposed mechanism, that is, TECH-CYCLES being applied to the instruction branch and TEMGA to the data branch.

We can see that, on average, the TC-HyOM approach achieves the best results in terms of total energy consumption considering twelve applications from Mibench. The hybrid mechanism achieves energy consumption savings of about 6%, 8%, 270%, and 252%, when compared with the TEMGA, TECH-CYCLES, TCaT, and TECH heuristics, respectively. More recently, approaches in the literature such as TECH-CYCLES and TEMGA have achieved the highest reductions when compared with TECH and TCaT mechanisms. The main difference between these two classes of

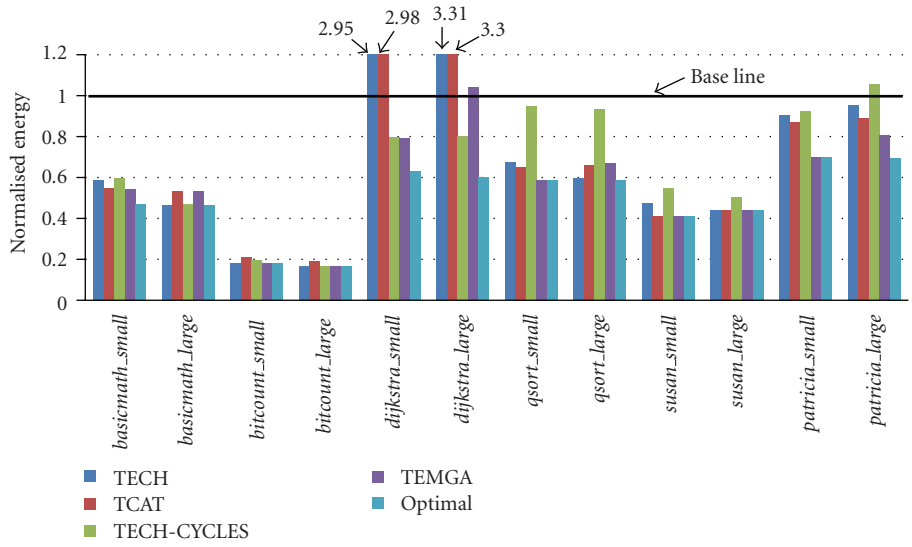

Figure 9: Energy consumption for data branch normalized to the base cache configuration for different optimization mechanisms (TECH, TCAT, TECH-CYCLES, and TEMGA) and optimal values.

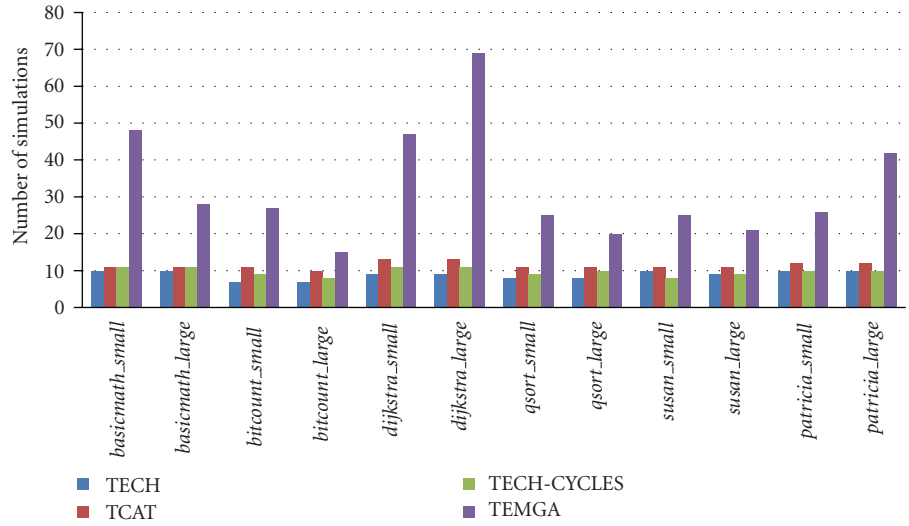

Figure 10: Number of simulations for data branch and for different optimization mechanisms.

optimization mechanism is the inclusion of the performance constraints in the objective functions. The use of these two parameters, that is, energy consumption and performance, can guide the heuristic mechanism to configurations with a lower energy consumption and processing time of the application.

The impact in terms of the number of simulations necessary to conclude the execution for each optimization mechanism was evaluated. Table 7 shows the number of simulations for 12 applications from the Mibench suite, considering the complete analysis for both branches (instruction and data) of the memory hierarchy. It was observed that TC-HyOM, on average, required about 42 simulations to conclude the simulation for both branches of the hierarchy.

We observe an average of about 17, 22, 19, and 86 simulations for the TCaT, TECH, TECH-CYCLES, and TEMGA heuristics, respectively, considering twelve applications from the Mibench benchmark and the analysis of both branches. Although the TC-HyOM mechanism is based on a genetic algorithm, specifically to a data branch, its hybrid approach reduces the total number of simulations when compared with a complete approach such as TEMGA, in which GA is applied in both branches. Reductions in the number of simulations for TC-HyOM can amount to 100%, when compared with the TEMGA approach.

The total energy consumption of the memory hierarchy was obtained for each optimization mechanism. Each column in Figure 11 represents the lowest energy consumption values of the hierarchy obtained for a given mechanism.

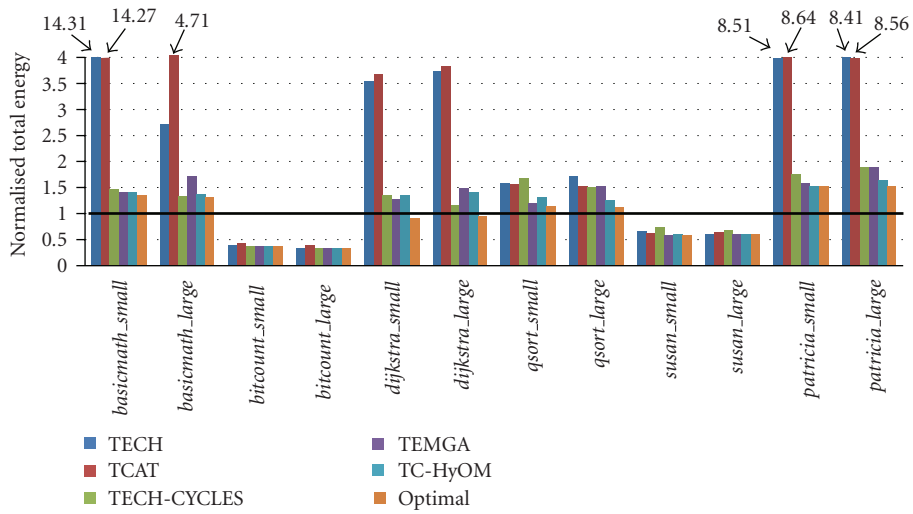

Figure 11: Total energy consumption for complete hierarchy normalized to the base cache configuration for different optimization mechanisms (TECH, TCAT, TECH-CYCLES, TEMGA, and TC-HyOM) and optimal values.

Table 6: Total Energy consumption for memory hierarchy normalized to base cache and compared with other heuristics.

Application (from Mibench suite)	TECH	TCAT	TECH-CYCLES	TEMGA	TC-HyOM
basicmath_small	14.3115	14.2722	1.4690	1.4181	1.4162
basicmath_large	2.7164	4.7177	1.3248	1.7124	1.3844
bitcount_small	0.3846	0.4386	0.3830	0.3698	0.3709
bitcount_large	0.3324	0.3848	0.3324	0.3315	0.3315
djkstra_small	3.5374	3.6716	1.3279	1.2703	1.3419
djkstra_large	3.7258	3.8201	1.1658	1.4813	1.4059
patricia_small	8.5120	8.6464	1.7637	1.5812	1.5354
patricia_large	8.4149	8.5626	1.8888	1.8836	1.6353
qsort_small	1.5805	1.5649	1.6764	1.1980	1.3171
qsort_large	1.7271	1.5249	1.5115	1.5217	1.2499
susan_small	0.6548	0.6208	0.7452	0.5896	0.6070
susan_large	0.6080	0.6346	0.6704	0.6093	0.6093
Average	3.8755	4.0716	1.1899	1.1639	1.1004
Sum	46.5055	48.8594	14.2788	13.9668	13.2048

Table 7: Number of simulations comparison for complete execution of different applications in both branches of the hierarchy considering different optimization mechanisms.

Application (from Mibench suite)	TECH	TCAT	TECH-CYCLES	TEMGA	TC-HyOM
basicmath_small	21	19	22	105	59
basicmath_large	24	20	22	73	39
bitcount_small	21	14	18	77	36
bitcount_large	20	14	17	37	24
djkstra_small	24	19	21	125	57
djkstra_large	23	18	20	96	78
patricia_small	22	17	20	165	36
patricia_large	22	18	20	74	52
qsort_small	21	17	19	50	35
qsort_large	21	17	19	74	29
susan_small	21	17	17	79	34
susan_large	21	16	17	81	29
Average	21.75	17.17	19.33	86.33	42.33

The TECH, TCaT, TECH-CYCLES, TEMGA, and TC-HyOM mechanisms were normalized to the same base cache in terms of energy consumption and evaluated for twelve applications from the Mibench suite.

In order to show the effectiveness of the proposed optimization mechanism (TC-HyOM) in energy consumption, we compared the proposed approach with optimal results. The results show that, on average, the proposed mechanism has configurations close to the optimal values, in some cases attaining the optimal values (patricia_small and bitcount_large). On average, TC-HyOM is only 11% less economic in terms of energy consumption than the optimal values.

6. Conclusion and Future Work

A hybrid optimization mechanism for cache hierarchy tuning considering energy consumption constraints was presented. This method (TC-HyOM) aggregates two different approaches for each branch of the hierarchy, namely, the instruction and data branches. The proposed mechanism determines a memory hierarchy that is only 11% less economic in terms of energy consumption than the optimal values and required, on average, about 42 simulations to conclude the optimization mechanism, representing about 9% of the design space.

Hybrid mechanisms may be indicated as a good alternative when both levels have separate instruction and data

caches because the optimization can be applied independently for the distinct branches. This kind of approach can optimize some design metrics in order to attain the best configurations compatible with the constraints of the system to be designed.

Some future work is being developed in a parallel study that is intended for two-level memory hierarchies with a unified second level. An alternative solution compared with a state-of-the-art such as ACE-AWT and SERP from [10] is being proposed. We have applied intelligent mechanisms, based on genetic algorithms, hybrid and multiobjective solutions to two-level hierarchies with a unified second level. In addition, other benchmarks are being incorporated in the next analysis and we plan to work with other platforms such as the ARM-based one.

Acknowledgments

This work was supported in part by CNPq (Universal 476839/2008-4), by FINEP (ref. 4950/06), and FACEPE, all Brazilian agencies.

References

[1] M. Kandemir and A. Choudhary, "Compiler-directed scratch pad memory hierarchy design and management," in *Proceedings of the 39th Design Automation Conference*, pp. 628–633, New Orleans, La, USA, June 2002.

[2] S. Wuytack, F. Catthoor, L. Nachtergaele, and H. De Man, "Power exploration for data dominated video applications," in *Proceedings of the International Symposium of Low-Power Electronics and Design (ISLPED '00)*, pp. 359–364, Monterey, Calif, USA, August 1996.

[3] C. Zhang and F. Vahid, "Cache configuration exploration on prototyping platforms," in *Proceedings of IEEE International Workshop on Rapid System Prototyping (RSP '03)*, pp. 164–170, June 2003.

[4] Altera, "NIOS Embedded Processor System," San Jose, Calif, USA, 2001, http://www.altera.com/products/ip/processors/nios/nio-index.html.

[5] ARM Processors Datasheets, "Access the complete manuals of all ARM processors," Cambridge, UK, 2010, http://infocenter.arm.com/help/index.jsp.

[6] MIPS Technologies Cores Processors, "MIPS Technologies Core Microprocessors for 32 and 64 bits," Mountain View, Calif, USA, http://www.mips.com/products/cores/.

[7] Tensilica, "Xtensa processor generator," Santa Clara, Calif, USA, http://www.tensilica.com/.

[8] C. Zhang, F. Vahid, and R. Lysecky, "A self-tuning cache architecture for embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 2, pp. 407–425, 2004.

[9] A. Gordon-Ross, F. Vahid, and N. Dutt, "Automatic tuning of two-level caches to embedded applications," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, pp. 208–213, February 2004.

[10] A. Gordon-Ross, F. Vahid, and N. D. Dutt, "Fast configurable-cache tuning with a unified second-level cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, Article ID 4689316, pp. 80–91, 2009.

[11] M. R. Guttaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: a free, commercially representative embedded benchmark suite," in *Proceedings of IEEE International Workshop on Workload Characterization (WWC '01)*, pp. 3–14, 2001.

[12] C. Zhang and F. Vahid, "Cache configuration exploration on prototyping platforms," in *Procceding of the 14th IEEE International Workshop on Rapid System Prototyping (RSP '03)*, p. 164, June 2003.

[13] A. G. Silva-Filho, M. E. Lima, P. S. B. Nascimento, and R. Eskinazi, "An energy-aware exploration approach based on open software enviroment," in *Proceedings of the International Embedded System Symposium (IESS '05)*, pp. 97–102, July 2005.

[14] A. G. Silva-Filho, F. R. Cordeiro, R. E. Sant'Anna, and M. E. Lima, "Heuristic for two-level cache hierarchy exploration considering energy consumption and performance," in *Proceedings of the International Workshop on Integrated Circuit and System Design, Power and Timing Modeling, Optimization and Simulation (PATMOS '06)*, pp. 75–83, September 2006.

[15] A. G. Silva-Filho, C. J. A. Bastos-Filho, R. M. F. Lima, D. M. A. Falcão, F. R. Cordeiro, and M. P. Lima, "An intelligent mechanism to explore a two-level cache hierarchy considering energy consumption and time performance," in *19th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD*, pp. 177–184, bra, October 2007.

[16] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," *Computer Architecture News*, vol. 25, no. 3, pp. 13–25, 1997.

[17] N. Dutt and M. Mamidipaka, "eCACTI: An Enhanced Power Estimation Model for On-chip Caches," TR 04-28, Set. 2004.

[18] (ITRS 2008), "International Technology Roadmap for Semiconductors 2008 (Update)," http://www.itrs.net/.

[19] S. Thoziyoor, N. Muralimanohar, and N. P. Jouppi, "HP Technical Reports: CACTI 5.0," May 2008, http://www.hpl.hp.com/techreports/2007/HPL-2007-167.html.

[20] T. Givargis and F. Vahid, "Platune: a tuning framework for system-on-a-chip platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1317–1327, 2002.

[21] Intel Datasheet Processors, http://download.intel.com/design/processor/datashts/318726.pdf.

[22] "Intel Datasheet Processors, Series 9300," http://download.intel.com/design/itanium/downloads/322821.pdf.