Journal of Systems Architecture xxx (2013) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Two-level caches tuning technique for energy consumption in reconfigurable embedded MPSoC

A. Bengueddach [a,b], B. Senouci [b], S. Niar [b,*], B. Beldjilali [a]

[a] University of Oran, Department of Computer Science, BP 1524, El-M'Naouer, Algeria
[b] University of Valenciennes Hainaut-Cambrésis, ISTV2 – Le Mont Houy, LAMIH-CNRS UMR, 59313 Valenciennes Cedex 9, France

ARTICLE INFO

Article history:
Received 1 May 2013
Received in revised form 20 May 2013
Accepted 26 May 2013
Available online xxxx

Keywords:
Embedded system
MPSoC
Cache memories
Reconfigurable architecture
Energy consumption
Optimization

ABSTRACT

In order to meet the ever-increasing computing requirement in the embedded market, multiprocessor chips were proposed as the best way out. In this work we investigate the energy consumption in these embedded MPSoC systems. One of the efficient solutions to reduce the energy consumption is to reconfigure the cache memories. This approach was applied for one cache level/one processor architecture, but has not yet been investigated for multiprocessor architecture with two level caches. The main contribution of this paper is to explore two level caches (L1/L2) multiprocessor architecture by estimating the energy consumption. Using a simulation platform, we first built a multiprocessor architecture, and then we propose a new algorithm that tunes the two-level cache memory hierarchy (L1 and L2). The tuning caches approach is based on three parameters: cache size, line size, and associativity. To find the best cache configuration, the application is divided into several execution intervals. And then, for each interval, we generate the best cache configuration. Finally, the approach is validated using a set of open source benchmarks; *Spec 2006*, *Splash-2*, *MediaBench* and we discuss the performance in terms of speedup and energy reduction.

1. Introduction

In order to meet the ever-increasing computing requirement in the embedded market, and in particular multimedia market; multiprocessor chips (MPSoC – Multiprocessor System on Chip) were proposed as the best way out. Game consoles and smart-phones are a typical example of these embedded MPSoC systems.

Several functionalities are integrated in such systems (e.g. voice communication, audio–video encoding, web browsing, exchange message, gaming, etc.). Thus, those functionalities need for a long lifetime battery and an effective power management on-chip technique.

In order to fill the gap between the CPUs speed and the global memory system, MPSoC architectures implements hierarchical memory structures (or *caches*). These memory hierarchies contribute largely in the energy consumption of the overall hardware/software architecture. Multilevel caches are responsible for a significant part. Over 50% of the total energy consumption system is due to its large on-chip area and high access frequency [1,2].

Customizing the memory hierarchy for a given application by reconfiguring the caches dynamically is being introduced as an efficient solution to save energy use in many processors based architectures. On the other hand a cache is organized by three parameters: the cache size (s), the cache line size (l), and the cache associativity (a).

Tuning the three cache parameters during the execution time to find the best cache configuration for a software task may considerably save the whole energy consumption of the system.

Dynamic cache reconfiguration has been well studied for single processor in both general-purpose computers as well as real-time embedded systems [3–8].

Typically, L2 cache acts as a shared resource in MPSoC. Recent research has shown that shared on-chip cache may become a performance bottleneck for MPSoC because of contentions among parallel running tasks [1,9,10].

Therefore to resolve this problem, adjusting a cache memory for a specific application implies to tune all these parameters for the different level of cache hierarchy, cache level 1 (L1), and cache level two (L2), to find the best cache configuration.

In this paper, we present a new heuristic that combines optimization techniques and considers the relation between tuning L1 and L2 cache memories in an MPSoC. On one hand caches reconfiguration based approach seems to be an efficient solution to reduce energy consumption in embedded MPSoC. On the other hand, it presents several challenges:

* Corresponding author. Tel.: +33 688496057.
E-mail addresses: smail.niar@univ-valenciennes.fr, smniar@gmail.com (S. Niar).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.05.018

- The three factors that affect the cache energy consumption and miss rates for a given application are the cache size, line size, and associatively [11,12]. The designer has to find the right cache configuration by setting these parameters, and reduce the energy consumption without affecting the system performance.
- Since the affiliation between L1 and L2 cache is central, the exploration process of two level caches should not be done separately. For example, a high L1 associativity decreases misses and thus reduces the need for large L2.
- By tuning cache parameters, L1 and L2 caches can be adjusted for a particular application. However, no unique cache configuration would be ideal for the whole application in term of energy consumption, since that the application can pass by different execution phases during its running lifetime. On one hand, to be efficient, the application should be divided into several execution intervals. On the other hand, predetermining real time operations for each interval becomes a hard issue to handle.
- Reconfiguring two level caches for a specific application by tuning different parameters (size, line, associativity), produce a huge design space exploration.

In this work we use the dynamic reconfiguration approach for the two caches (L1 and L2) to reduce the energy consumption in a multiprocessor architecture. We first build a symmetric multiprocessor architecture with a shared L2 cache. Once the architecture is built, we propose a new algorithm to find the best cache configurations for the target application regarding energy consumption without greatly impacting the execution time.

The rest of the paper is organized as follows: Section 2 gives an overview of related works. Section 3 gives an insight of cache reconfiguration methodology. Section 4 describes the experimentations and presents some results. Section 5 concludes the paper.

2. Related work

Reconfigurable computing is considered as a solution for HW/SW systems design in order to reduce the power consumption in such systems.

Malik et al. [2] introduce caches reconfiguration approach. They present M•CORE M340 processor, which contains an 8-Kbyte, 4-way set-associative unified L1 cache with 16-byte line size. The cache sub-system supports programmable modes, which allow certain features of the cache to be enabled/disabled for power and performance tuning.

In [5] authors proposed a configurable cache when the number of configurations is small. Exhaustive search methods may be used to find optimal cache configurations, but the time required for an exhaustive search could be very important for complex and modern MPSoC.

Several tools were developed for assisting designers in tuning a single level cache. Platune [13] is a framework for adjusting configurable SoC. It utilizes the exhaustive search method for only one-level cache; due to the exploration space, which is very large.

The same work, as [13], is presented by Palesi et al. in [15] with the possibility to reduce the configuration space by using a genetic algorithm.

Zhang's configuration method in [11,12,16] is based on the importance of the cache parameters: cache size, line size and associativity. In this approach, they tune two of the cache parameters and fixed the third one. Each configuration is analyzed in terms of miss rates and energy consumption.

The proposed method considers only one-level cache. A. Gordon-Ross extended the initial heuristic, producing the *Two-Level Cache Tuner* (TCaT) heuristic [6]. It consists in exploring a small percentage of the configuration space, taking into account a two-level cache hierarchy. The methodology analyzes also the impact of each parameter in terms of energy and number of cycles spent for a given application. The methodology analyzes also the impact of each parameter in terms of energy and number of cycles spent for a given application. However, the heuristic consider only single processor architecture. The *Two-level Cache Exploration Heuristic considering CYCLES* (TECH-CYCLES) heuristic [8] considers the impact of energy consumption to determine if a solution is better than another, it also considers the execution time of an application. In practical terms, the exploration of a given cache parameter in TECH-CYCLES continues while it is possible to optimize energy consumption and execution time.

Ref. [1] presents a novel energy optimization technique for multiprocessor system, which efficiently integrates dynamic reconfiguration of private caches and partitioning of the shared L2 cache in the granularity of ways. Each core is assigned a limited number of ways in the cache and will only access that portion in all cache sets. In Table 1 we summarize the key differences between the existed configuration techniques and our approach.

Finally, the tuning heuristic Conditional Parameter Adjustment Cache Tuner (CPACT) presented in [14] explores level one data cache (L1) in a heterogeneous dual-core system, where each data cache can have a different configuration. However, this approach didn't consider the configuration of second level cache.

Ref. [17] presents a L1 data cache tuning heuristic for a heterogeneous multi-core system, which classifies applications based on data sharing and cache behavior, and uses this classification to guide cache tuning and reduce the number of cores that need to be tuned. However both [14] and [17] explore the design space without considering L1/L2 dependency. Thus, cache tuning reveals substantial energy savings for single processor with two level cache, and multiprocessor with one level cache, but has not yet be investigated for multiprocessor architecture with two level cache, which is the aim of this paper.

3. Caches reconfiguration methodology

In this section, we present the different steps of our methodology of adjusting caches for a specific MPSoC application illustrated by Fig. 1.

Based on the Multi2Sim framework [18], we first built a multiprocessor architecture with two levels of cache memories. This

Table 1
Comparison techniques.

Techniques	Principle	Constraints	Fast	Complexity
DCR + CP algorithm [1]	Reconfiguring L1 with partitioning L2 + multiprocessor	Energy + cycles	++	++
TCaT [6]	2 Level cache + single processor	Energy + cycles	++	
TECH-cycles [8]	2 Level cache hierarchy + single processor	Energy + cycles	++	++
Zhang's heuristic [11]	Only one level cache + one processor	Energy	+++	+
CPACT [14]	1 Level cache + multiprocessor	Energy	+++	+++
Our approach	2 Level cache + multiprocessor	Energy + cycles	+++	++

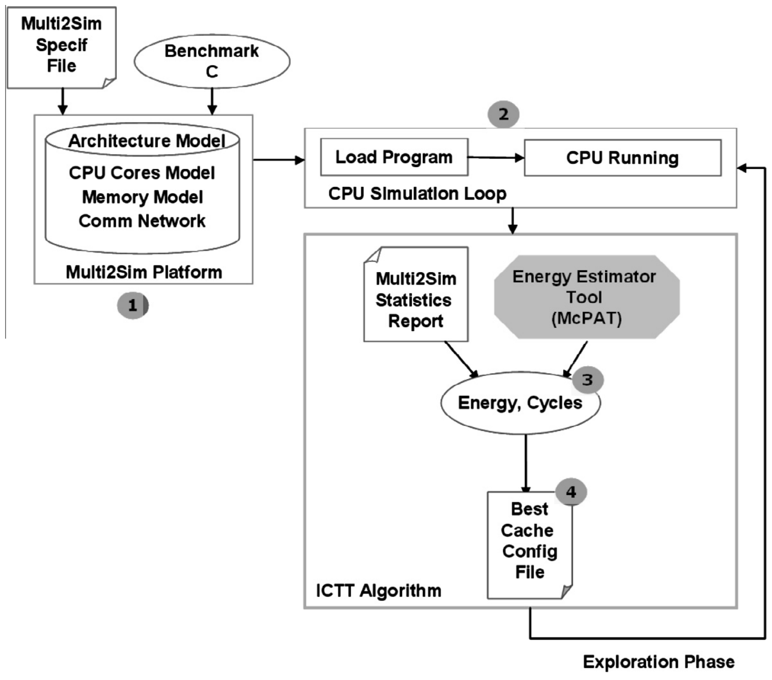


Fig. 1. Cache Reconfiguration Flow Diagram. Step 1: Input Multi2Sim: Specification File + Benchmark C. Step 2: Statistic report generating. Step 3: Estimation of the energy consumption by MCPAT tool. Step 4: Exploration using modified ICTT algorithm, and automatic generation of the best cache configuration.

step consists mainly in writing the Multi2Sim specification file and developing applications to run on the simulator.

The second step is to determine the best hardware configuration with the best size, line, and associativity for a given architecture. This step generates also a statistic report after a simulation loop of several cache configurations that will be the entry point of the McPAT energy estimation tool [19].

Then, the *Interval Caches Tuning Technique* (ICTT) algorithm explores the design space of configurations in order to find the best and the appropriate cache size, line size, and associativity in term of execution time and energy consumption for the given application. Details about these steps are given in the following paragraphs.

3.1. Hardware simulated architecture model

Multi2Sim is a simulation platform for HW/SW systems. It allows an easy definition of parameterized system architecture. Several hardware components are described in the Multi2Sim library. In such platform, three main parts of the architecture are defined: the processor(s)/core(s), the memory hierarchy (caches and main memory), and the interconnection network.

The inputs of Multi2Sim are the application benchmark written in C and a specification file for the hardware architecture. In our case the specification file includes dual-core architecture.

Fig. 2 shows a typical architecture with private L1 cache instruction *il1* and data cache *dl1* with a shared L2 cache. Those caches are highly reconfigurable in terms of cache size, line size and associativity. Our architecture model is based on the multithreading using an internal system scheduling allowed by the Multi2Sim. The scheduler maps the software threads on the processing nodes.

Fig. 3 summarizes the specification of the Multi2Sim based architecture. The sections *[CacheGeometry<name>]* define cache geometries that is, a set of characteristics used to create the L1 and L2 caches, respectively.

The next two sections *[Net<name>]*, not shown in Fig. 3, define the interconnection networks. In this case, two of them are created, called *net-0* and *net-1*, respectively. Both interconnects are defined

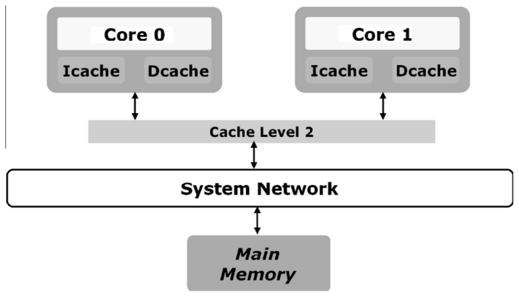


Fig. 2. Multi2Sim based MPSoC architecture.

```
[ CacheGeometry l1 ]
Sets = 128
Assoc = 2
BlockSize = 32
Latency = 2
Policy = LRU
[ CacheGeometry l2geo ]
Sets = 512
Assoc = 4
BlockSize = 32
Latency = 20
[ Net net-0 ]
Topology = Bus
LinkWidth = 32
[ Net net-1 ]
Topology = BUS
LinkWidth = 32
[ Cache dl1-0 ]
Geometry = l1
LoNet = net-0
[ Cache dl1-1 ]
Geometry = l1
LoNet = net-0
[ Cache l2 ]
Geometry = l2geo
HiNet = net-0
LoNet = net-1
[ Cache il1-0 ]
Geometry = l1
LoNet = net-0
[ Cache il1-1 ]
Geometry = l1
LoNet = net-0
[ MainMemory ]
HiNet = net-1
BlockSize = 64
Latency = 200
[ Node Node 0/0 ]
Core = 0
Thread = 0
ICache = il1-0
DCache = dl1-0
[ Node Node 1/0 ]
Core = 1
Thread = 0
ICache = il1-1
DCache = dl1-1
```

Fig. 3. Multi2Sim specification file.

with a bus topology and a link width of 32 bytes.

The section entitled *[Cache<name>]* creates caches, and the section *[MainMemory]* in the Multi2Sim specification file defines the main memory parameters. The main memory is connected to network *net-1*, and has 64-byte as block size.

Finally, section *[Node<name>]* defines the processing node. Each node corresponds to thread executed on specific core; and caches assigned by the variables DCache and ICache.

3.2. McPAT – Multi2Sim: automatic adaptation

Multi2Sim has been adapted to provide statistics that McPAT requires in its input file. However, the models provided by McPAT and Multi2Sim are different. Fig. 4 shows, the script written to automate the edition of McPAT's input file (Fig. 5).

The code in Fig. 5 illustrates how an input file to McPAT is created. As mentioned, data from the statistic report are stored into the script to recuperate the number of each parameter in L1 data cache.

3.3. Problem overview

We define the problem of reconfiguration of the caches regarding energy consumption for a given application [20]. We distinguish two kinds of applications: Data-stream applications, such as in multimedia, where the data arrive continually and the application runs for a long period of time. And, Constant applications that are executed for a known period of time and with a known in advance number of instructions to execute. In the case of constant applications, we divide the running applications into "*n*" fixed intervals. Currently "*n*" is determined through experimentation. This value depends on the applications to be executed and could be changed according to desired degree of reconfiguration. We consider here the following variables:

- $A = \{I_1, I_2 \ldots I_n\}$ where A is the application divided into "n" tuning intervals with fixed number of instructions per interval.
- The architecture configuration set $C = \{c_1, c_2 \ldots c_m\}$, where "$m$" is the number of possible configurations for L1 and L2 caches.
- The reconfiguration R is the consumed energy to switch from the configuration Ck to any other configuration Cl ($k \neq l$). In this paper, we assume that the reconfiguration time is constant regardless Ck and Cl.

```
/*Start Energy Calculation Procedure*/
void energy_estimation() {

/*Variables Declaration*/
FILE* archi_model, cpu_report, caches_report, stat_report;
FILE* input_mcpat, output_mcpat;
int cycles;
double power, energy;

/*Generate the report statistics*/
concatenate(cpu_report, caches_report, stat_report) ;

/*Generate McPAT's input file*/
input_mcpat = input_mcpat(stat_report, archi_model);

/*Call for McPAT Tool*/
system(. / mcpat inputmcpat > mcpat_out);

/*Extracting Cycles and Power Output File*/
cycles = extract("Cycles", mcpat_out);
power = extract("RuntimeDynamic", mcpat_out);

/*Energy Calculation*/
energy = energy_estimation(Cycles, power);
} /*End Energy Calculation Procedure*/
```

Fig. 4. Energy calculation procedure.

```
/*Start Data Cache Specification*/

<component id="system.core0.dcache" name="dcache">

/*Cache geometry and cache controller buffer sizes*/

<param name="dcache_config"
value="8192,16,4,1,1,3,16,0"/>
<param name="buffer_sizes" value="16, 16, 16, 16"/>

/*Read accesses, read misses, and cache line conflicts carried out
previously on the top of Multi2Sim */

<stat name="read_accesses" value="200000"/>
<stat name="write_accesses" value="27276"/>
<stat name="read_misses" value="1632"/>
<stat name="write_misses" value="183"/>

</component> /*End Data Cache Specification*/
```

Fig. 5. McPAT input file.

- The execution consists of a sequence of intervals S, such as $S = <I_1, I_2, I_3, I_4, I_5, I_6,\ldots>$. Where $S[k]$ is the interval tuning at position "k" in the sequence S. In the above sequence, $S[1]$ is I_1, $S[2]$ is I_2, and $S[3]$ is I_3.

The total energy consumption E_{total} of the application depends on the interval's energy consumption with the corresponding configuration (see Eq. 1), and the time needed for the design space reconfiguration.

$$E_{total} = \sum_{i=1}^{n} Eli \text{ such as } Ii \in A \quad (1)$$

In order to reduce the total energy consumption, one of the difficult assignments is to find the best configuration for every application's interval. We also define the architecture reconfiguration problem for data stream applications, which processes data as soon as they arrived. In our approach, tuning interval is based on current and past, but not future application behavior. In this case the number of tuning intervals varies, and when the final configuration is determined, the application is executed in that final configuration for the rest of the application's execution.

3.4. Energy model

The energy consumption for multi-level cache sub-system is modeled as the sum of energy consumption of the L1 cache and the shared L2 cache, denoted by $E_{L1}(s_1, l_1, a_1)$ and $E_{L2}(s_2, l_2, a_2)$ respectively as shown in Eq. 2:

$$E_{li} = E_{L1}(s_1, l_1, a_1) + E_{L2}(s_2, l_2, a_2) \quad (2)$$

Our goal is to find the cache parameters "s", "l" and "a" such that the overall energy consumption E_{total} of the cache subsystem is minimized. Let P_{L1} denote the power consumption of cache L1, and P_{L2} denote the power consumption of cache L2. E_{L1} and E_{L2} are simply computed using (3), respectively as:

$$\begin{cases} E(L1) = P(L1) * Cycles \\ E(L2) = P(L2) * Cycles \end{cases} \quad (3)$$

where *Cycles* represent the execution time required for one interval executed with two level caches L1 and L2. The value of P_{L1} and P_{L2} are calculated using CACTI tool [19], and based on Eq. (4).

$$\begin{cases} lP_{(L1)} = P_{\text{Dynamic}(L1)} + P_{\text{Short\_circuit}(L1)} + P_{\text{Leakage}(L1)} \\ P_{(L2)} = P_{\text{Dynamic}(L2)} + P_{\text{Short\_circuit}(L2)} + P_{\text{Leakage}(L2)} \end{cases} \quad (4)$$

The first term is the dynamic power, used up by the capacitive loads when the circuit switches states. The second term represent the short-circuit power consumed when both the pull-up and pull-down devices in a CMOS circuit are partially on. The third term is the static power consumed by the transistors.

3.5. Approach and methodology

In this work we use the dynamic reconfiguration approach for two-level caches in multiprocessor architecture. Based on a software platform, we first build a multiprocessor architecture with two processor-cache levels. Once the architecture is built, we define three parameters for cache reconfiguration (cache size, line size, and associativity), and then we develop a new algorithm to generate the suitable cache configuration for the target application considering energy reduction. Our dynamic approach solves the problem of reconfiguration of the caches for an embedded application.

We combine two approaches: CPACT and TECH-Cycles. The first one considers L1 for two processors; and the second one explores the two levels for one processor. The following paragraphs describe the different steps of our heuristic.

So, the ICTT heuristic described considers two level caches hierarchy with several parameters (s_2: level two cache size, s_1: level one cache size, l_2: level two cache line size, l_1: level one cache line size, a_2: level two cache associativity and a_1: level one cache associativity). Minimum and maximum values (MIN, MAX) for these parameters define the configuration space.

ICTT initialize all the caches parameters with MIN values, and then the first vector of parameters is defined. The six-positions indicate the current value of each parameter as illustrated in Fig. 7. The parameters of the vector select from the left to the right. The best cache size is obtained for the two-level cache by following the four steps of the heuristic, as shown in Fig. 6.

a. *Step 1: Initial Step:* In this step, we tune the first parameter (s_2) of the initial vector from MIN to MAX by power of 2. The order is important to guarantee the correct use of the heuristic. While we tune one parameter, we keep the other ones fixed. For example, during size tuning, the heuristic starts with a 128 Kbytes cache L2 (Fig. 8); the line size and associativity are fixed at their smallest values while the cache size is explored from MIN to MAX. Then, we gradually increase the total cache size to our largest possible size as long as increasing the size of the cache results in a decrease in total energy. We continue this process until we find the best (s_2) in term of energy consumption, or we reach the MAX tune of (s_2). The line size and associativity are tuned in a similar way.
b. *Step 2:* In this step, the heuristic performs a size adjustment task. This adjustment starts by tuning the second level cache size (s_2), and then the first cache size (s_1). Since the cache size has the largest impact on energy consumption, this second step tries to tune just the cache size without the other parameters (l, a). We mention that this tuning goes through the smallest values of the cache size.
c. *Step 3:* Condition 1 evaluates the energy results of the size adjustment step. The best cache size is obtained from the two-level cache by varying its size (s_2) decreasingly. We

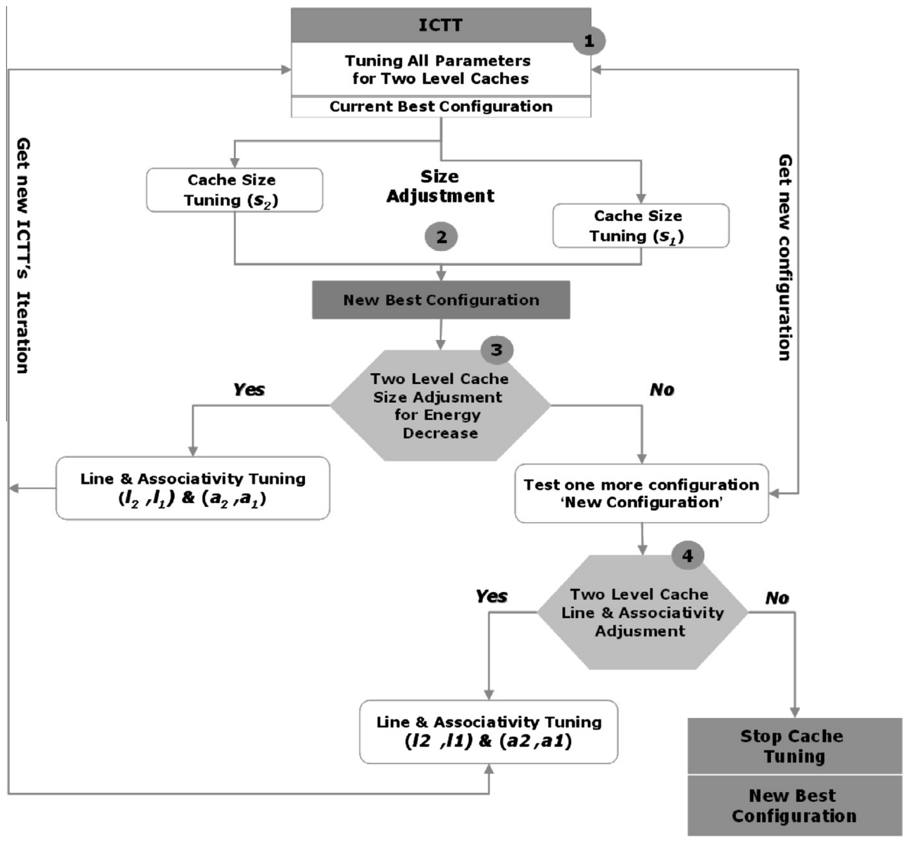

ICTT
1
Tuning All Parameters for Two Level Caches
Current Best Configuration
Cache Size Tuning (s_2)
Size Adjustment
Cache Size Tuning (s_1)
2
New Best Configuration
3
Two Level Cache Size Adjusment for Energy Decrease
Yes
No
Line & Associativity Tuning (l_2, l_1) & (a_2, a_1)
Test one more configuration 'New Configuration'
4
Yes
Two Level Cache Line & Associativity Adjusment
No
Line & Associativity Tuning (l2 ,l1) & (a2 ,a1)
Stop Cache Tuning
New Best Configuration
Get new ICTT's Iteration
Get new configuration

Fig. 6. ICTT heuristic steps. Step 1: Applies ICTT heuristic for L1/L2 cache parameters. Step 2: Performs a size adjustment task. Step 3: Evaluates the energy results of the size adjustment step. Step 4: Gets the best configuration for L1/L2 cache.

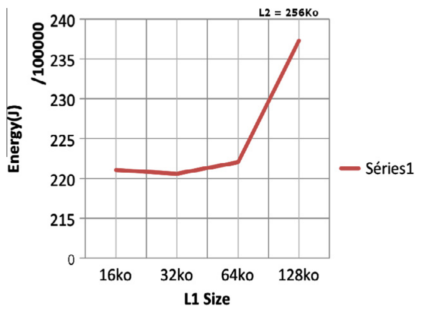


Fig. 9. L1 cache energy dissipation (*Sha* Appli).

More we increase the cache size; the energy consumption increase; this due to the number of cache access that also increases. In the same time, the cache size directly affects miss rate. Increasing cache size involves reducing of cache miss rate and vice versa. However, increasing the cache size will not improve performance greatly but will increase energy dissipation significantly. It is very important to keep the near-memory cache miss rate as low as possible because it directly influences memory energy consumption.

This first experiment has as main goal to check the hardware simulation architecture in term of memory hierarchy. In the second series of experiments we apply our two level cache tuning heuristic to find the best cache configuration for a constant application.

4.3. ICTT results and analysis

In this section, we apply the ICTT algorithm on two different categories of application: constant application and data stream one. We considered just the *Basicmath* application as case study to explain in details all aspects of the heuristic.

4.3.1. ICTT: constant applications

In embedded systems, the application could pass by different phases during its running lifetime. In order to be efficient in each phase of the application, we divided the application into several intervals, and then we find the best cache configuration for each interval in term of energy use. Choosing the right interval tuning is important. Since the tuning process consumes extra energy which represents the cost of the reconfiguration process in term of consumed energy.

If the tuning interval is big, optimal configurations can be skipped. Then, we observed that is convenient to start the tuning by precise value. Based on our experience we set the interval value by 500,000 cycles.

Thus, another parameter is taken into account during the exploration process. This parameter refers to the number of iterations (Design Space Exploration) being explored inside the same interval. The number of iterations range from 1 to 486 as shown in Eq. (5).

In constant application with identified number of instructions, the ICTT explores several iterations before converges to the “one” that will be used for the rest of the instructions inside the same interval. Thus, the heuristic tunes the cache parameters for each execution phase. To simulate interval tuning “*I*” in Multi2sim, we changed every 500,000 cycles the cache configuration until the ICTT steps are done. ICTT set the caches parameters in the beginning of each interval. Fig. 10 represents the best cache configuration found for each “*I*” in term of energy use. We mention that in this example the number of the intervals is fixed. This can be illustrated in the application *Basicmath_large*, which pass by different phases and then need for different cache configurations {C_{485}, C_0, C_{161}, ... C_3}. For example, C_{162} (configuration number 162) is the best cache configuration for I_3 (see Fig. 11).

A red curve represents the energy consumption of the application *Basicmath_large* during one interval tuning, and the blue one represents the number of cycles in this same interval. We note that for I_3 our heuristic explores a number of configurations to converge to the best solution.

Here we consider that the best configuration corresponds to the optimization of both energy and execution time in the same time. The both constraints have the same parity. Resulting in an average energy reduction of 1% within the phase interval. Similarly, the improvement of performance cycles CPU is 1% in the interval tuning.

The graph in Fig. 12 represents the explored configurations by the heuristic during its execution time for the *Basicmath_large* application. Given that, it is only from the configuration number C_{432} the heuristic starts to have a better solution in terms of execution time and energy. The best configuration was number C_{485}.

In this example, and to avoid the local optimal solution, which corresponds to the peak at point C_{161}, the heuristic (Fig. 6, step 3) tries to diversify the exploration space allowing a bad solution. As we can see in Fig. 12, the heuristic check twice the best configuration (C_{485} in this case).

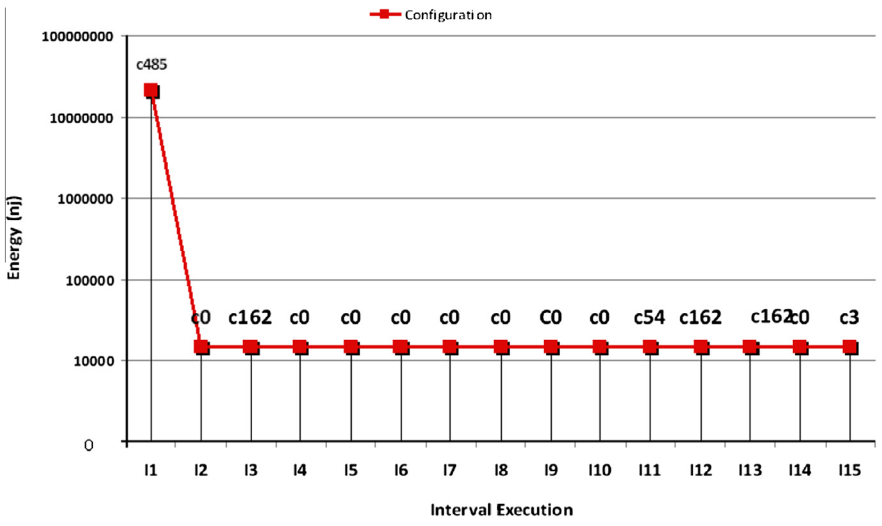


Fig. 10. ICTT: constant application.

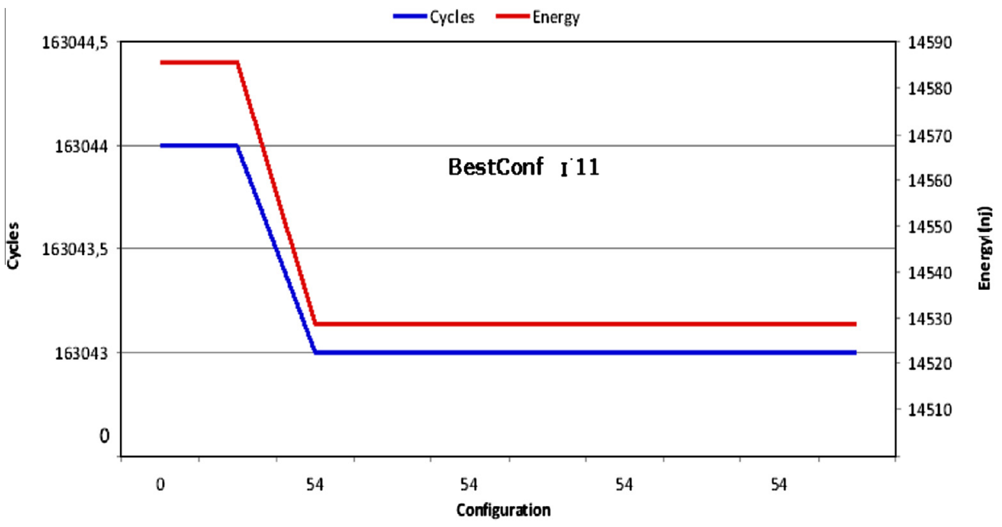


Fig. 14. ICTT's convergence behavior in I_{11}.

Table 4
The number of configurations explored by Interval.

Interval tuning (I)	Configuration number (NB_Cfg)
I1	19
I2	8
I3	10
I4	8
I5	8
I6	8
I7	8
I8	8
I9	8
I10	8
I11	10
I12	10
I13	10
I14	8
I15	9

configurations are eliminated during the tuning of caches parameters when the tuning way switch from MIN to MAX and form MAX to MIN in a row as shown in Fig. 8.

In Table 4, we show the several configurations corresponding to ICTT exploration by each interval tuning considering *Basicmath_large*. As we can see, the results indicated that, on average the proposed heuristic does not exceed 8 configurations out of 486 possible configurations equal to 1% of exploration space.

4.3.2. ICTT: data stream application

In the case of data stream applications, the reconfiguration decision is taken with incomplete information about the upcoming data. Then, in data stream case, the ICTT heuristic manage the cache reconfiguration in a different way. In fact, the cache reconfiguration is done just for a defined number of iterations and then we force the heuristic to converge and get the best cache configuration for given interval. This configuration is used for the rest of the execution time (all the rest of the intervals). In other words, the heuristic is applied just for few intervals and then it converges for all the rest of the application.

Fig. 15 shows while ICTT determined the final configuration; the benchmark is executed in that final configuration for the rest of the benchmark's execution.

These experiments prove the efficiency of ICTT algorithm in term of convergence speed up. So, in these applications case

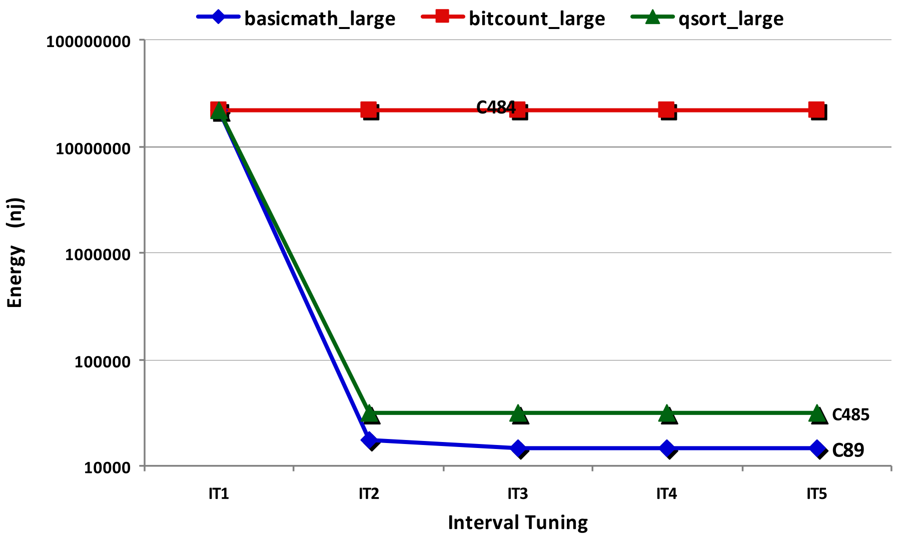


Fig. 15. ICTT: Data stream applications.

Table 5
Cache parameters example.

Cache parameters best Cfg/App	*S2*	*S1*	*L2*	*L1*	*A2*	*A1*
C 484/bitcount	64	8	64	64	4	2
C 485/qsort	64	8	64	64	4	4
C 89/basicmath	16	4	64	16	4	4

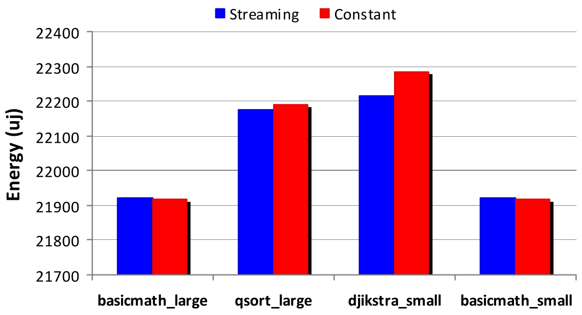


Fig. 16. Energy consumption (in uJ) for *constant and data-stream modes.*

studies (Fig. 15), two tuning phases were sufficient to find the best caches configuration. For example the *Bitcount* application converges on the C_{484}, also *Qsort* converge on C_{485}. Except the *Basicmath* application, where the optimal configuration was C_{89}, the heuristic converge after five tuning intervals. Table 5 illustrates the cache parameters corresponding to those configurations.

4.4. Comparison's studies

For comparison purpose, we choose a reference/base cache configuration for L1 and L2, that defined respectively by (8KB, 64B line size, and 64KB, 4-way), and (64KB, 64B line size, 4-way). Then we calculate energy saving for a given configuration, by normalizing the system's energy consumption with base cache configuration as shown in Eq. (6):

$$\text{Normalized Energy} = \frac{\text{Energy Heuristic}}{\text{Energy Base Cache}} \qquad (6)$$

In the upcoming sections we will present various comparisons of our heuristic via different experiments. For all experiments, we keep the previous experimental environment.

4.4.1. Constant and stream data application

In this section we describe a comparison in energy consumption between applications based data flow instruction and applications with constant behavior. The graphs in Fig. 16 illustrate this comparison in term of energy use during the execution time. A red column represents the ICTT tuning for constant applications, and the blue one represents the ICTT tuning for the data-stream execution.

In fact this figure shows that the energy used in the case of application with constant behavior is more significant and this is mainly due to initialization phase in the starting of each interval. This phase of initialization is performed just once in the case of applications with data-stream instructions.

4.4.2. Optimal approach and ICTT tuning

In order to demonstrate the efficiency of our approach in reducing MPSoC's energy consumption, we compare our results with the optimal approach (exhaustive research). Then, we calculate the energy consumed in the case of an exhaustive research, and the energy consumed in the case of ICTT solution. Fig. 17 clarifies this comparison using 9 applications (benchmarks).

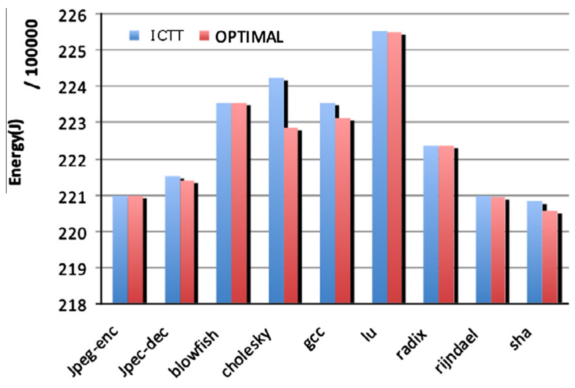


Fig. 17. Energy (in J/10,000) comparison: ICTT vs. Optimal solution.

We can see that the optimal solution is found by the ICTT heuristic for the most benchmarks. In two analyzed benchmarks (*Jpeg-enc*, *Blowfish*) ICTT reaches the lowest energy consumption results. Moreover, the modified ICTT reduces the configuration space exploration significantly. The exhaustive approach for a two level cache hierarchy explores 64 cache configurations. The improved heuristic explores only 10 cache configurations, which correspond to 15% of the space exploration [21]. Moreover, we find the same solution as the exhaustive approach. The reduction in the configuration space exploration speeds up both a simulation.

5. Conclusion

In this paper we evaluate the energy consumption in embedded MPSoC system using cache-tuning approach. It concerns two level caches (L1/L2) in multiprocessor architecture. The cache tuning was applied for one processor based architecture with one cache level, but has not yet been studied for multiprocessor architecture with two level caches.

We propose an efficient solution to reduce the energy use in cache memories. The proposed solution is based on tuning the cache memories for a given application that have been divided into several intervals. The tuning is based on three parameters: cache size, line size, and associativity.

In the proposed framework, the ICTT reconfiguration algorithm took into account two types of applications, with constant instructions, and data-flow.

In the first one, the heuristic explores and locates the best cache configuration for each interval in term of energy saves, and generates automatically the best cache configuration parameters for each interval of the application. In that way the total energy consumption for a given application is reduced.

But for applications with continued data-stream, the reconfiguration decision is taken with incomplete information about the upcoming data. In that case, the ICTT heuristic accomplishes the cache reconfiguration in a different way.

Therefore, the cache reconfiguration is done just for a defined number of iterations and then we force the heuristic to converge and get the best cache configuration for given interval. This configuration is used for the rest of the execution time of the application.

Finally, the approach was validated using several benchmarks and we discuss the performance in terms of energy reduction. We have also to mention that the heuristic is more suitable for an application with variable tuning. And more appropriate if we need a rapid exploration to get a solution near to optimal for our architecture design.

Furthermore, future work will focus on implementing our approach of caches memories configuration at RTL (HDL) level using a FPGA platform and an open source multiprocessor architecture (Sparc LEON).

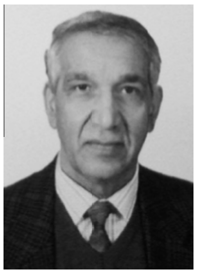
References

[1] W. Wang, P. Mishra, S. Ranka, Dynamic cache reconfiguration and partitioning for energy optimization in real-time multicore systems DAC 2011, June 5–10, 2011, San Diego, California, USA, 2011.
[2] A. Malik, W. Moyer, D. Cermak, A low power unified cache architecture providing power and performance flexibility, in: Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED 2000.
[3] A. Gordon-Ross, F. Vahid, N. Dutt, Fast configurable-cache tuning with a unified second-level cache, in: Proceedings of the ISLPED 2005 International Symposium on Low Power Electronics and Design.
[4] A. Gordon-Ross, P. Viana, F. Vahid, W. Najjar, E. Barros, A one-shot configurable-cache tuner for improved energy and performance, Design, Automation & Test in Europe Conference & Exhibition (2007) 1–6.
[5] D.H. Albonesi, Selective cache ways on-demand cache resource allocation, Journal of Instruction Level Parallelism, May 2000.
[6] A. Gordon-Ross, F. Vahid, N. Dutt, Automatic tuning of two-level caches to embedded applications, in: Proceedings of the Conference on Design, Automation and Test in, Europe – DATE 2004.
[7] D. Benitez, J.C. Moure, D.I. Rexachs, E. Luque, A reconfigurable data cache for adaptive processors, ARC2006, Springer-Verlag, LNCS3985, 2006, pp. 230–242.
[8] A.G. Silva-Filho, F.R. Cordeiro, R.E. Sant Anna, M.E. Lima, Heuristic for two-level cache hierarchy exploration considering energy consumption and performance, PATMOS 2006, LNCS 4148, 2006, pp. 75–83.
[9] R. Reddy, P. Petrov, Eliminating inter-process cache interference through cache reconfigurability for real-time and low-power embedded multi-tasking systems, CASES, 2007.
[10] D. Kaseridis et al., Bank-aware dynamic cache partitioning for multicore architectures, ICPP 2009.
[11] C. Zhang, F. Vahid, W. Najjar, A highly-configurable cache architecture for embedded systems, in: Proceedings of ISCA 2000, the 30th Annual International Symposium on Computer Architecture.
[12] C. Zhang, F. Vahid, Cache configuration exploration on prototyping platforms, in: 14th IEEE International Workshop on Rapid System Prototyping, June 2003, vol. 00, p. 164.
[13] T. Givargis, F. Vahid, Platune: a tuning framework for system-on-a-chip platforms, IEEE Transactions on Computer-Aided Design 21 (2002) 1–11.
[14] M. Rawlins, A. Gordon-Ross, CPACT – The conditional parameter adjustment cache tuner for dual-core architectures, ICCD 2011, 396–403.
[15] M. Palesi, T. Givargis, Multi-objective design space exploration using genetic algorithms, International Workshop on HW/SW Codesign, May 2002.
[16] C. Zhang, F. Vahid, R. Lysecky, A self-tuning cache architecture for embedded systems, ACM Transactions on Embedded Computing Systems 3 (2) (2004) 407–425.
[17] M. Rawlins and A. Gordon-Ross. An application classification guided cache tuning heuristic for multi-core architectures, ASP-DAC 2012.
[18] R. Ubal, J. Sahuquillo, S. Petit, Pedro Lopez, Z. Chen, D.R. Kaeli, The Multi2Sim simulation framework computer architecture and high performance computing, SBAC-PAD 2007, in: 19th International Symposium on Computer Architecture and High Performance Computing.
[19] S. Li, J.H. Ahn, J.B. Brockman, N.P. Joupp, McPAT 1.0: An Integrated Power, Area, and Timing Modeling Framework for Multicore Architectures, <http://www.hpl.hp.com/research/mcpat/>.
[20] C. Huang, D. Sheldon, F. Vahid, Dynamic Tuning of Configurable Architectures: The AWW, Online Algorithm, CODES+ISSS 2008.
[21] A. Bengueddach, B. Senouci, S. Niar, B. Beldjilali, Energy consumption in reconfigurable MPSoC architecture: two-level caches optimization oriented approach, in: International Design and Test Symposium (IDT'12), December14–15, Qatar, 2012.
[22] A.G. Silva-Filho, F.R. Cordeiro, A combined optimization method for tuning two-level memory hierarchy considering energy consumption, EURASIP Journal on Embedded Systems 2011.

Asmaa Bengueddach received her engineer degree in Computer Science from the University of Oran Es-Senia, Algeria in 2004. Currently she is working as assistant professor and preparing her PhD thesis within the computer science department in the University of Oran Es-Senia Algeria. Her research interests include embedded multiprocessor systems design and reconfigurable computing.

Benaoumeur Senouci received his Ph.D in computer engineering from the National Polytechnic Institute of Grenoble-TIMA Laboratory (France) in 2008; currently he is working as PostDoc in CNRS/University of Valenciennes with LAMIH laboratory. Before joining the CNRS, he was working as research follow in the university of Twente (The Netherlands). His research topics includes platform based embedded HW/SW systems design, and dependability of embedded systems.

Smail Niar (University of Valenciennes & CNRS, France) received his PhD in Computer Engineering from the University of Lille in 1990. Since then, he has been professor at the University of Valenciennes. He works in the "Mobile & Embedded Systems" research group at the "Laboratory for Automation, Mechanical and Computer Engineering", a joint research unit between CNRS and the University of Valenciennes.



Bouziane Beldjilali received his PhD degree in Computer Science from the University of Algiers in 1996. Now, he is working as professor at the University of Oran Es-Senia, Algeria. He works as the head of Computer Science Departement till 2004. Also, he is leading a research group in LIO Laboratory at the University of Oran Es-Senia. His research interests include embedded multiprocessor systems design and Production Systems.