Research Article

An ESL Approach for Energy Consumption Analysis of Cache Memories in SoC Platforms

Abel G. Silva-Filho, Filipe R. Cordeiro, Cristiano C. Araújo, Adriano Sarmento, Millena Gomes, Edna Barros, and Manoel E. Lima

Informatics Center, Federal University of Pernambuco, Avenida. Jornalista Aníbal Fernandes, s/n, Cidade. Universitária, 50740-560 Recife, PE, Brazil

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

Received 1 June 2010; Accepted 10 November 2010

Academic Editor: Elías Todorovich

Copyright © 2011 Abel G. Silva-Filho et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The design of complex circuits as SoCs presents two great challenges to designers. One is the speeding up of system functionality modeling and the second is the implementation of the system in an architecture that meets performance and power consumption requirements. Thus, developing new high-level specification mechanisms for the reduction of the design effort with automatic architecture exploration is a necessity. This paper proposes an Electronic-System-Level (ESL) approach for system modeling and cache energy consumption analysis of SoCs called PCacheEnergyAnalyzer. It uses as entry a high-level UML-2.0 profile model of the system and it generates a simulation model of a multicore platform that can be analyzed for cache tuning. PCacheEnergyAnalyzer performs static/dynamic energy consumption analysis of caches on platforms that may have different processors. Architecture exploration is achieved by letting designers choose different processors for platform generation and different mechanisms for cache optimization. PCacheEnergyAnalyzer has been validated with several applications of Mibench, Mediabench, and PowerStone benchmarks, and results show that it provides analysis with reduced simulation effort.

1. Introduction

The design of integrated circuits respecting time-to-market has become a challenge for designers. One important aspect that must be taken into consideration is the number and complexity of the functionalities the circuit must implement, including communication between the functionalities. It is difficult for designers to model the functional intent of the specification in existing low-level languages like C and VHDL/Verilog that are commonly used for the implementation of the system. Descriptions tend to be lengthy and hard to maintain. Errors on the implementation are usually hard to find and even harder to correct, thus raising further the design effort. A significant effort is being done to reduce design time by providing methodologies and tools to raise the abstraction level of system modeling. Despite these approaches are commonly known as Electronic System Level (ESL) [1], they differ in the language and abstraction levels supported. For instance,

communication modeling levels may vary from lower to higher abstraction capabilities: *register transfer level*, *driver level*, *message level*, and *service level* [2]. Despite this effort methodologies and tools that work on service level are still missing.

A second aspect to be observed is design space exploration. Although SoCs have become an alternative as target architecture for complex circuit implementation, their use implies that parts of the circuit functionality are implemented as software applications running on processors and the rest as hardware components. This means that a large design space should be considered in the mapping of system functionality in the implementation platform. A SoC provides different configurations for designers, each one presenting a different power consumption and performance. Currently, the energy consumed by memory hierarchies can account for up to 50% of the total energy spent by microprocessor-based architectures [3]. Moreover, static memory consumption cannot be neglected anymore.

Currently it accounts for about 40% of the energy consumption in CMOS circuits [4, 5]. This can vary for distinct embedded memory structures. It is, therefore, of vital importance to try to estimate the impact of memory hierarchy consumption prior to system implementation taking into consideration both dynamic and static aspects. In fact, many approaches do not take both aspects in consideration. Previous researches have observed that adjustment of the cache parameters to a specific application can save energy consumption in the memory system [6–9]; thus, many efforts have been made to save energy through optimizations in cache memories. However, since the main goal of cache subsystems is to provide high performance for memory access, cache optimization techniques are supposed to be driven not only by energy consumption reduction but also by preventing degradation of the application performance. There is no single combination of cache parameters (total size, line size and associativity), also known as cache configuration, which is suitable for every application. Therefore, cache subsystems have been customized in order to deal with specific characteristics and to optimize their energy consumption when running a particular application. By adjusting the parameters of a cache memory to a specific application, it is possible to save an average of 60% of energy consumption [6]. Nevertheless, finding a suitable cache configuration for a specific application on a SoC can be a complex task and may take a long simulation and analysis time. Most of the current approaches that use exhaustive or heuristics-based exploration [7–9] are costly and time consuming.

This paper describes a design flow with tool support for the service level modeling of SoCs using a proposed service-level UML-ESL profile and for Design Space Exploration (DSE) of cache configuration aiming to semiautomatically select a suitable performance/energy consumption tradeoff for the target platform. Differently from other approaches it takes into consideration several processors in the SoC and dynamic and static memory energy consumption. It includes a fast exploration strategy based on single-step simulation for simulating multiple sizes of caches simultaneously and is not bound to a specific processor. All these features are integrated in an easy-to-handle graphical environment in the PDesigner Framework. In fact, this work extends the approach proposed in [10], by providing now an environment that (i) allows designers to model systems in ESL (Electronic System Level) [1] abstraction level with automatic platform generation, reducing thus platform modeling time, and (ii) supports different cache analysis techniques, giving designers the freedom to vary cache analysis precision level and simulation time depending on the target application.

The rest of this paper is structured as follows. In the next section, we discuss some recent related work. Section 3 presents the proposed mechanism for ESL system modeling and a cache energy analysis approach. In Section 4 some experiments on several applications are presented comparing the cache analysis results for two different processors and two different cache analysis techniques by using the proposed approach. Finally, Section 5 presents conclusions and future directions.

2. Related Work

This section discusses related work on the domains of high-level modeling and design space exploration based on cache configuration. As far as the authors are concerned there is no approach that takes into account these two aspects together. The need for high-level specification mechanisms has led to the development of domain-specific specializations of UML 2.0 profiles. Examples of such specializations are the UML profile for SoC [11] that maps SystemC-based constructs to UML. MARTE [12] is a UML profile for real time and embedded systems that provide support for specification, design, and verification/validation development stages. Despite enabling high-level modeling, this approach neither provides a path to generate the high-level model to a SoC nor performs any cache analysis. Mueller et al. [13] proposed a UML 2.0 profile for SystemC based on the UML profile for SoC and a tool that generates SystemC code from UML diagrams. Mueller also demonstrates how to model hardware and software together in a unique UML-based approach. Riccobene and Scandurra [14] proposed an approach for hardware and software codesign using UML and SystemC. Its tool employs a UML platform-independent model (PIM) that can be mapped onto a platform-specific model (PSM), which contains design and implementation details. However, in both approaches, UML designs are merely representations of SystemC models. Designers must model ports, interfaces, and protocols, which means that low-level implementation details still have to be modeled. Furthermore, both approaches do not cover cache analysis.

Regarding cache-based design space exploration some existing methods still apply traditional exhaustive search to find the optimal cache configuration in the design exploration space [15]. That means that for each cache configuration a simulation must be run. However, the time required for such alternative is often prohibitive. One example is the Platune framework [16] that uses an exhaustive search method for selecting a one-level cache configuration in SoCs. This approach is also limited by the fact that it supports only the MIPS processor. Palesi and Givargis [17] reduce the possible configuration space by using a genetic algorithm and produces faster results than the Platune approach. Zhang and Vahid [6] have developed a heuristic based on the influence of each cache parameter (cache size, line size, and associativity) in the overall energy consumption. The simulation mechanisms used by the previous approaches are based on the SimpleScalar simulator [18] and CACTI tools [19, 20] and are limited to dynamic energy analysis. Prete et al. [21] proposed the simulation tool called ChARM for tuning ARM-based embedded systems that also include cache memories. This tool provides a parametric, trace-driven simulation for tuning system configuration. Unlike the previous approaches, it provides a graphical interface that allows designers to configure the components parameters, to evaluate execution time, to conduct a set of simulations, and to analyze the results. However, energy consumption is not supported by this approach. On the other hand, Silva-Filho et al. in [9], take into account static and dynamic energy consumption estimates in the heuristic analysis approach called

TECH-CYCLES. This heuristic uses the eCACTI [20] cache memory model to determine the energy consumption of the memory hierarchy. This is an analytical cache memory model that has been validated using HSPICE simulation tools. The eCACTI, differently from CACTI-based approaches, considers the two energy components: static and dynamic, thus being a more accurate model. PDesigner [22] is a framework based on Eclipse [23] that provides support for the modeling and simulation of SoCs and MPSoCs virtual platforms. By using this framework the platform designer can build platforms using a graphical interface and generate an executable platform simulation model. PDesigner is a free solution and offers support for platform modeling with different components such as processors, cache memory, memory, bus, and connections. The framework also allows performance evaluation; however, energy results are not supported.

Looking at the situation depicted in Table 1 there becomes evident the lack of a mechanism that provides facilities for modeling a system at a high abstraction level and for generating and analyzing multiple platforms with caches. The analysis mechanism should be able to estimate both dynamic and static energy consumption of cache memories and should be efficient enough (single pass simulation) for supporting design space exploration of cache architectures. The proposed work is designed to support all the features shown in Table 1.

3. The Proposed Approach

In this paper, an ESL approach for SoC modeling and architecture exploration based on cache energy consumption estimation is proposed. This proposed approach provides three main benefits. First, it reduces the design effort by supporting the description of a hardware/software (HS) system at service level, using a domain specific UML-ESL profile. Second, it automates the generation of a SoC architecture that is optimized regarding static and dynamic cache energy consumption. Third, the tools implementing these techniques have been integrated as a plugin in the PDesigner framework.

The proposed approach is depicted in Figure 1. The system is specified using the UML-ESL profile. This profile supports the description of hardware and software modules that are connected by a “use” relationship. That means that software modules use services provided by hardware modules. Each service call is split in transactions that execute in a specific order determined in the description. The partitioning and transactions ordering are then extracted from the original description for further processing. This includes the mapping of the partitioning onto a base platform, which includes processors, memories, and interconnection structure. The base platform is composed of elements stored in a component library and does not contain yet hardware and software support for communication between the components. The next step is the automatic mapping of the transactions ordering from the original description to the base platform resulting in an initial virtual platform

that implements the UML-ESL specification. Once this initial SoC architecture is created efficient, techniques including heuristics and single pass simulation are applied in order to generate intermediary platforms and estimate performance and static and dynamic energy consumption of the cache configurations. This cycle is repeated until there is no need of tuning the cache parameters. In the remaining part of this section, the input flow depicted in Figure 1 is discussed in more detail.

3.1. UML-ESL Profile and Platform Generation Flow. This section discusses the UML-ESL profile and how it reduces the modeling effort of HS systems and how this system specification is automatically mapped onto a SoC target. Using UML-ESL the designer must first specify the HS architectural partitioning system. The specification also includes the usage relationship between each partitioned module in the system. Each usage relationship means that one module is able to call services provided by another module. A service call is composed of transactions and in the scope of this work a transaction is defined as the execution of a service, from its request to its response, where a response may include a return value. Transactions are classified according to the categories proposed by Gogniat et al. [24] and summarized in Table 2. Regarding type, transactions can be synchronous or asynchronous. In a synchronous transaction the service instance is available for just one call at a time, remaining blocked until communication is completed. On the other hand, in an asynchronous transaction the service instance is available for more than one request; that is, it does not need to finish the execution of a service to deal with another transaction. The communication mode corresponds to the sequence in which the services are executed and is classified as sequential or parallel.

In the architecture specification the system is partitioned in hardware and software modules, which are represented by classes in the class diagram. Each module is formed by attributes and operations. In the UML-ESL profile services are identified as operations that can be public or visible to external modules. Partitioning is supported by classifying the modules in hardware or software ones. The classification is performed using the stereotypes *sw_module* and *hw_module*. Figure 2 shows an example of architectural modeling using the UML-ESL profile. It is composed of one software module, *Skin*, and a hardware module *Mahalanobis*. The usage relationship between modules is specified by directed connection with the “<< use >>” arrow. This specifies that module *Skin* is able to use the public service *mahalanobisDistance()* provided by the *Mahalanobis* hardware module. An optional notation at each end of the “<< use >>” arrow indicates the multiplicity of instances of that module as presented.

The service-based communication is specified using sequence diagrams, where the designer is able to define whether the service call will be synchronous or asynchronous by using different arrowheads. Filled arrowheads specify synchronous calls while asynchronous calls use stick arrowheads. In Figure 3, the call to *service1()* is an example of a synchronous call. The service call ordering is specified by the order in which it appears considering the vertical axis,

TABLE 1: Comparison of related studies.

	Multiplatform modeling	Single pass simulation	Dynamic consumption	Static consumption	Graphical exploration	ESL abstraction level*
Zhang	—	—		—	—	—
Palesis	—	—		—	—	—
Silva-Filho	—	—			—	—
Platune	—	—		—		—
SimpleScalar	—	—	—	—	—	—
ChARM	—	—	—	—		—
PDesigner		—	—	—		
MARTE	—	—	—	—		—
Muller & Riccobene	—	—	—	—

FIGURE 1: Proposed Approach.

which is an abstract time representation. In the sequential diagram the designer can define the communication mode to be sequential or parallel using frame *nameboxes*. Using the parallel *namebox* the designer specifies that each service call in the frame represents a thread of execution performed in parallel. A service call can be executed more than once. The

number of times executed can be set and made explicit by inserting the corresponding number of service call instances in the sequence diagram. When the number of execution is not set, a service call or a set of service calls are grouped in a frame with the text “loop” in the frame’s namebox, indicating the repetition of the set of calls.

FIGURE 2: Example of architectural partitioning in UML-ESL.

FIGURE 3: Tracking sequence diagram with parallel communication.

TABLE 2: Communication classification summary table.

Characteristic	Classification	
Type	Synchronous	Asynchronous
Mode	Sequential	Parallel

3.1.1. System Level Intermediate Format (SLIF). In this work there was developed the System Level Intermediate Format (SLIF) that supports a service level representation of a system and may be used for further processing by other design tools. The SLIF format and how it is used are depicted in Figure 4. A parser was developed that takes the system description in UML-ESL (upper part of Figure 4) and generates a tree representation (middle part of the Figure 4). The initial node of the tree, *system*, represents the entire design. The following nodes *system architecture* and *transactions ordering* represent the partitioning of the system and communication interactions, respectively. Under the system architecture node it can be seen the nodes representing the type (hardware or software), eventual parameters, and services provided by the module and modules whose services are used by this one. Each parameter has associated to it a name that uniquely

identifies it and a data type node. For every service node there are, respectively, nodes for name and input/output data types nodes. The *used services* node contains the services where the module is able to call from other modules. *UsedModule* node contains the name of the modules that provide the requested services as well as the name of the requested services. Each set of transactions that has a sequential or parallel execution order is grouped in one *Transaction Ordering*. This node is composed of a communication mode (sequential or parallel), and transactions. Each transaction has an order, which indicates the execution order of the service. A transaction has also a type (synchronous or asynchronous), a service, which must be a reference to a previously identified service of a system module, and a caller, that must be a reference to the module that requests the service.

The lower part of Figure 4 shows the architecture of the generated platform using the SLIF as entry. White boxes represent high-level models of the hardware/software modules. Those modules are mapped onto a base platform composed of processor(s), bus and memory. Black boxes represent these base platform elements, stored in a platform library. The automatic generated communication infrastructure is highlighted in gray and includes device drivers, device

FIGURE 4: Service level system representation and target virtual platform.

controllers, interfaces, and communication controller. The input flow is the architectural and transactional information of the system represented in the SLIF format described previously. For each interconnection between the processor and the bus a cache analyzer component is associated. This component will be used in the next phase for the energy consumption estimation.

3.2. Cache Energy Consumption Analysis Flow. Figure 5 shows the flow to analyze energy consumption in SoCs cache memories. Designers select a set of properties for each cache analyzer in order to define the design exploration space, transistor technology, and type of exploration to be performed. The design exploration space is configured by the definition of minimum and maximum values for each cache

TABLE 3: Energy evaluation for two applications (*bitcount_small*, *patricia_small*).

(a) Bitcount_small.						
SimpleScalar				ArchC		
Conf	Miss	MEM	Energy	Miss	MEM	Energy
(2048, 32, 1)	0.03%	380288	0.00256	0.01%	109184	$2.09E - 03$
(2048, 32, 2)	0.02%	280256	0.003752	0.01%	111072	$3.55E - 03$
(2048, 32, 4)	0.02%	214176	0.005191	0.01%	111296	$5.19E - 03$

(b) Patricia_small.						
SimpleScalar				ArchC		
Conf	Miss	MEM	Energy	Miss	MEM	Energy
(2048, 32, 1)	19.67%	$8.40E + 08$	1.734637	5.58%	$518E + 08$	1.077279
(2048, 32, 2)	13.57%	$5.79E + 08$	1.202464	5.43%	$5.04E + 08$	1.057696
(2048, 32, 4)	10.31%	$4.4E + 08$	0.920564	5.02%	$4.65E + 08$	0.98929

FIGURE 5: Energy consumption analysis flow.

memory parameter. The parameters are cache size, cache line size, and associativity degree. Transistor technology is defined by selecting a value in nanometers among available technologies. Further, the type of exploration mechanism that can be either an exhaustive approach or the available heuristics is chosen. An exhaustive approach uses the single pass simulation technique proposed in [25]; it generates misses and hits statistics for the entire configuration space defined by the designer. So, the result of the simulator execution is an XML file that contains the cache configuration ID, cache parameters such as size, line size, associativity

FIGURE 6: Energy calculation flow.

degree, number of accesses, and miss rate for all cache configurations. The advantage of using the single pass is performance as it can be 70 times faster than traditional exhaustive simulation-based mechanism for ADPCM application from Mediabench benchmark [26]. One limitation of this approach is that it can be applied to only one level of cache hierarchy.

The exploration space may contain cache configurations that are invalid or that are not interesting for the designer. After simulation, the designer is able to select some or all configurations for energy analysis and define the configuration space that contains all the desired cache configurations through a Configuration Selection Window. After the configuration space has been defined, the energy module calculates the energy consumption and the number of cycles for each selected configuration. For a heuristic-based exploration, the configuration is initially set as heuristic approach and an executable simulation model of the platform is generated. For that configuration, energy and cycle values are obtained and results are evaluated. Based on the configuration results,

FIGURE 7: Energy estimation interactive chart.

FIGURE 8: Normalized energy comparison for SimpleScalar and PCEA.

FIGURE 9: Energy estimation for different applications.

if the selected heuristic has not ended, a new configuration is set and the process is repeated until the heuristic stops.

The cache memory energy consumption calculation flow is depicted in Figure 6. A parser receives as input the selected

FIGURE 10: Zhang heuristic versus exhaustive approach for *bitcount* application.

configuration space saved in the XML file and separates it in two pieces of information. The first one is the cache parameter and technology information that is provided for the eCACTI tool to calculate the dynamic and static energy per access. The second one contains the number of misses, the number of accesses, and cache parameters of the chosen configuration. This information, along with the dynamic and static energy provided by the eCACTI, is used to calculate the total static and dynamic energies consumed by the cache memory for the application. In addition, in this step, the total number of cycles necessary to execute the application is also calculated. Once these parameters have been calculated, another parser generates the energy estimation results for each configuration also in XML format file.

The total number of cycles and total energy estimations are calculated based on the set of equations described as follows. The miss rate is calculated following (1), dividing the number of misses (*Number_Miss*) by the number of cache memory accesses (*Number_Access_Cache*). The number of cycles due to misses in the cache is calculated based on (2), multiplying the miss rate (*Miss_Rate*) by the number of accesses (*Number_Access_Cache*) and by the number of cycles

FIGURE 11: Total Energy consumption normalized to the base cache configuration for Zhang Heuristic and Optimal values.

due to one miss in the cache (*Penalty*). The last value is calculated based on the cache line size in bytes. The total contribution of cycles due to the cache memory is calculated following (3), adding the number of cycles due to the misses in the cache (*Number_Cycles_Miss_Cache*) and the number of accesses in the cache (*Number_Access_Cache*)

$$\text{Miss_Rate} = \frac{\text{Number_Miss}}{\text{Number_Access_Cache}}, \quad (1)$$

$$\text{Number_Cycles_Miss_Cache} = \text{Miss_Rate} * \text{Number_Access_Cache} * \text{Penalty}; \quad (2)$$

$$\text{Total_Cycles_Cache} = \text{Number_Cycles_Miss_Cache} + \text{Number_Access_Cache}. \quad (3)$$

Additionally, the total energy consumption is calculated based on the set of equations described as follows. Equation (4) shows the number of words read from main memory and is calculated by multiplying the number of misses of the cache (*Number_Miss_Cache*) by the cache line size in bytes (*Cache_Line_Size*). The energy consumption due to the access to main memory is calculated based on (5). It is 3.1x the static and dynamic energy consumption per access the cache memory [4]. Thus, the total energy due to the main memory accesses is calculated through (6), multiplying the number of words read from main memory (*Words_Read_MEM*) by the energy per access to main memory (*Energy_MEM_per_Access*). The internal main memory model considers a similar approach adopted by [5], which is based on a low power 64-Mbit SDRAM using 0.18 μm CMOS technology manufactured by Samsung as reference. Following, the energy consumed by a cache memory described in (7) is calculated by adding individual contributions of dynamic and static energy and

processor stalls. Finally, the total energy consumption result (*Total_Energy*) is given by (8):

$$\text{Words_Read_MEM} = \text{Number_Miss_Cache} * \text{Cache_Line_Size}; \quad (4)$$

$$\text{Energy_MEM_per_Access} = 3.1 * (\text{Eleak_per_Access} + \text{Edyn_per_Access}); \quad (5)$$

$$\text{Energy_MEM} = \text{Words_Read_MEM} * \text{Energy_MEM_per_Access}; \quad (6)$$

$$\begin{aligned} \text{Energy_Cache} &= (\text{Number_Access_Cache} * \text{Edyn_per_Access}) \\ &+ (\text{Total_Cycles_Cache} * \text{Eleak_per_Access}) \\ &+ (\text{Number_Cycles_Cache} * \text{Energy_Stall}); \end{aligned} \quad (7)$$

$$\text{Total_Energy} = \text{Energy_MEM} + \text{Energy_Cache}. \quad (8)$$

A cost function given by the equation $F = \text{Energy} \times \text{Cycles}$ is also calculated. The minimization of this cost function makes it possible to keep the cache configurations near to Pareto-optimal [17]. These cache settings represent a tradeoff between performance and energy consumption. The configuration with the lowest $\text{Energy} \times \text{Cycles}$ cost is also identified. Once the energy calculation flow is concluded, the user can visualize the results of the cache energy analysis.

The energy consumption estimation for each configuration in the selected design space is displayed in a visual interactive chart as depicted in Figure 7. The chart displays on the y-axis the energy consumed in Joules and on the x-axis the performance in number of clock cycles. Each point on the chart corresponds to one of the configurations in the design space. The chart is interactive; so the user can select one of the points and see more information about it. There are two types of information: the first one, in the form of

a tool tip, is depicted by the rectangle in Figure 7 and contains the number of cycles and energy consumed by the selected configuration; the second form of presenting information is by viewing properties, also shown in Figure 7. The displayed cache information includes the cache configuration parameter values, miss rate, number of accesses, the cost value based on the cost function calculation, dynamic and static energy consumption, the total cycles required to run the application, and the total energy consumption. The configuration with the lowest calculated cost is represented in the interactive chart in a different color. The user can use this configuration as a reference to continue exploring the design space. He or she can also interact with the chart in order to view the properties of a particular cache configuration. In this step, the designer selects one of the configurations that best meets the performance/energy consumption requirements by simply clicking on the point in the chart. In this step, the designer updates the platform by replacing the cache analyzer with the actual cache component with the selected parameter values. The plugin replaces the cache model automatically by interacting with the PDesigner Framework.

4. Experimental Results

The results obtained with the proposed approach (PCache Energy Analyzer or PCEA) were compared with SimpleScalar by using the *basicmath_small* application from Mibench suite [27] and the same set of cache configurations. A design space with cache size varying from 256 to 8192 bytes was used, cache line size ranging from 16 to 64 bytes, and the associativity degree varying from 1 to 4, in a total of 50 different configurations. The selected technology was 0.18 μm , although it is not limited to this value as PCache Analyzer may be configured to work with other technologies. Although SimpleScalar does not support energy consumption analysis, it was calculated with an approach based on the work of Zhang and Vahid [6], using one level cache and the eCACTI cache memory energy model. Energy consumption was analyzed for different cache configurations. Each point in Figure 8 represents the energy consumption for a given cache configuration (cache size, cache line size, associativity).

Results shown in Figures 8 and 9 indicate that proposed approach presents a 100% fidelity when compared to SimpleScalar even when the behavior of the applications is different. In the first case in Figure 8, the nature of the applications results in a decrease in energy consumption with an increase in the associativity degree. This is not always true, as the reader can observe, considering a second comparison with SimpleScalar using the *bitcount_small* and *patricia_small* applications from Mibench suite benchmark. Results in Tables 3(a) and 3(b) for the *bitcount_small* application show that an increase in the associativity degree implies in the increase of energy consumption. Note that in this case, due to the low temporal locality of the application, there is not a significant reduction of main memory accesses. On the other hand, the *patricia_small* application has a different behavior. In this case, the increase in the associativity degree leads to a significant miss rate reduction. This feature, associated

with the substantial decrease in main memory accesses, helps to reduce the energy consumption of the application even with an increased associativity degree. Once again, despite the fact that absolute results are different for SimpleScalar and the proposed approach, the results obtained exhibit high fidelity, which is essential to explore the design space.

In order to show the possibility of using the proposed approach with different processors, Figure 9 summarizes the energy consumption estimation results for a cache configuration that optimize the cost function and for a configuration with the lowest energy consumption for each application and considering MIPS and SPARCV8 processors. Although these two processors have similar architectures, compilers and compilation optimization can introduce some differences in both scenarios. Figure 9 shows that the MIPS processor presented a reduction of 50% in energy consumption comparing with the SPARCV8 processor for the application *timing* from Mediabench benchmark suite [26]. Additionally, PCacheEnergyAnalyzer was employed to explore cache memory design space for three different applications of the Mediabench: *fft*, *rawcaudio*, and *rawdaudio*. In these applications the MIPS processor presented similar energy consumption than the SPARCV8 processor.

The third feature of the proposed approach is to provide a more efficient design space exploration. This has been achieved by integrating the heuristic proposed by Zhang and Vahid [6] in the proposed approach. The efficiency in the design space exploration was evaluated by using three applications of Mibench benchmark: *fft*, *bitcount*, and *basicmath_small*, and eleven applications of the PowerStone benchmark [28]: *adpcm*, *bcnt*, *blit*, *crc*, *engine*, *fir*, *g3fax*, *jpeg*, *pocsg*, *qurt*, and *ucbqsort*.

The design space exploration considered 32 different configurations for each application with the following parameters: cache sizes ranging from 1024 up to 8192 bytes, line sizes from 16 up to 64 bytes, and the associativity degree ranging from 1 to 4. Figure 10 shows the results obtained with the proposed approach in comparison to an exhaustive approach for the *bitcount* application. The rectangles in the chart represent the configurations simulated by the tool and show that the best results that fall into Pareto Optimal area have been simulated. This means that only 31% of the original exploration space was needed for the simulation. Similar results have been found for the other applications.

The graph in Figure 11 shows better results in the use of the Zhang heuristic when taking into account optimal results in terms of energy consumption. The energy consumption was normalized to a *base cache*: with cache size of 1024, line size of 16, and associativity degree of 4.

Figure 11 shows that best energy consumption solutions of Zhang Heuristic were close (>93% accurate) to optimal values in most cases. This concludes that in most cases, the proposed approach is able to find configurations near to optimal values with a lower exploration effort. In a similar way, other heuristics can be easily added to PCacheEnergyAnalyzer therefore providing a better energy consumption analysis and improvements in the platform architecture.

5. Conclusions and Future Work

This paper presented an ESL-based mechanism for modeling and energy consumption analysis of SoCs. The proposed approach combines a set of techniques and mechanisms that reduce the designer effort in the specification phase and, consequently, the number of errors. This is achieved by the use of the UML-ESL profile, a constrained UML 2.0 profile along with an automatic mapping flow to a predefined virtual platform. This profile reduces the platform modeling effort by taking as input a high abstraction level model of the system, which hides platform low-level implementation details, and automatically generates a virtual SoC which can be simulated and analyzed further.

Additionally, the proposed approach provides efficient and flexible support for cache memory energy consumption estimation of SoC virtual platforms. On one hand it allows the choice between efficient heuristics or single pass simulation in order to explore the cache design space in the platform. So the user is able to choose the one that presents better results. On the other hand it supports different processors and bus models for platform generation, thus yielding to a broader and more effective design space exploration.

Eighteen applications from Mibench, Mediabench, and PowerStone benchmarks were applied in order to validate the proposed approach in terms of fidelity and efficiency. Initial studies were focused for one level caches, however; it can be easily extended to more levels. Results have shown that it is a powerful mechanism to help users to find efficient cache configurations for a specific application, which considers not only performance but also the best relation between performance and energy consumption.

The PCacheEnergyAnalyzer approach fills the gaps of the existing approaches by simultaneously providing a high-level platform modeling mechanism, multiplatform support, extensibility, dynamic and static energy consumption estimation, and a graphic interface. We are currently working on the development of architecture exploration mechanism based on evolutionary algorithms.

Acknowledgments

This work was supported in part by CNPq (476839/2008-4 and 309089/2007-7) and by FINEP (ref. 4950/06), both Brazilian agencies.

References

- [1] G. Martin, B. Bailey, and A. Piziali, *ESL Design and Verification: A Prescription for Electronic System Level Methodology*, Morgan Kaufmann, San Francisco, Calif, USA, 2007.
- [2] K. Svarstad, G. Nicolescu, and A. Jerraya, "A model for describing communication between aggregate objects in the specification and design of embedded systems," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '04)*, pp. 77–85, Munich, Germany, March 2001.
- [3] R. Banakar, S. Steinke, B. S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: a design alternative for cache on-chip memory in embedded systems," in *Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES '02)*, pp. 73–78, Estes Park, Colo, USA, May 2002.
- [4] M. Mamidipaka, K. Khouri, N. Dutt, and M. Abadir, "Analytical models for leakage power estimation of memory array structures," in *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '04)*, pp. 146–151, September 2004.
- [5] W. Zhang, M. Kandemir, M. Karakoy, and G. Chen, "Reducing data cache leakage energy using a compiler-based approach," *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 3, pp. 652–678, 2005.
- [6] C. Zhang and F. Vahid, "Cache configuration exploration on prototyping platforms," in *Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping*, p. 164, June 2003.
- [7] A. Gordon-Ross, F. Vahid, and N. Dutt, "Automatic tuning of two-level caches to embedded applications," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '04)*, pp. 208–213, February 2004.
- [8] A. Gordon-Ross, F. Vahid, and N. Dutt, "Fast configurable-cache tuning with a unified second-level cache," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '05)*, pp. 323–326, August 2005.
- [9] 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 16th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS '06)*, vol. 4148 of *Lecture Notes in Computer Science*, pp. 75–83, Montpellier, France, September 2006.
- [10] F. R. Cordeiro, A. G. Silva-Filho, C. C. Araujo, M. Gomes, E. N. S. Barros, and M. E. Lima, "An environment for energy consumption analysis of cache memories in SoC platforms," in *Proceedings of the 6th Southern Programmable Logic Conference (SPL '10)*, pp. 35–40, March 2010.
- [11] "UML Profile for SoC, OMG," http://www.omg.org/technology/documents/formal/profile_soc.htm.
- [12] F. Mallet, C. André, and J. DeAntoni, "Executing AADL models with UML/MARTE," in *Proceedings of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '09)*, pp. 371–376, June 2009.
- [13] W. Mueller, A. Rosti, S. Bocchio et al., "UML for ESL design—basic principles, tools, and applications," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD '06)*, pp. 73–80, San Jose, Calif, USA, November 2006.
- [14] E. Riccobene and P. Scandurra, "Weaving executability into UML class models at PIM level," in *Proceedings of the 1st Workshop on Behaviour Modelling in Model-Driven Architecture*, June 2009.
- [15] L. Benini, A. Macii, and M. Poncino, "Energy-aware design of embedded memories: a survey of technologies, architectures, and optimization techniques," *ACM Transactions on Embedded Computing Systems*, vol. 2, no. 1, pp. 5–32, 2003.
- [16] 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.
- [17] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES '02)*, pp. 67–72, May 2002.

- [18] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," *Computer Architecture News*, vol. 25, no. 3, pp. 13–25, 1997.
- [19] P. Shivakumar and N. P. Jouppi, "Cacti 3.0: an integrated cache timing, power and area model," WRL Research Report 2001/2, 2001.
- [20] N. Dutt and M. Mamidipaka, "eCACTI: an enhanced power estimation model for on-chip caches," Tech. Rep. TR 04-28, September 2004.
- [21] C. A. Prete, M. Graziano, and F. Lazzarini, "The ChARM tool for tuning embedded systems," *IEEE Micro*, vol. 17, no. 4, pp. 67–76, 1997.
- [22] M. Gomes, C. C. Araujo, E. Barros, S. Rigo, R. Jardim, and G. Araujo, "Platform designer: an approach for modeling multiprocessor platforms based on SystemC," *Journal on Design Automation for Embedded Systems*, vol. 10, no. 4, pp. 253–283, 2007.
- [23] Eclipse, <http://www.eclipse.org/>.
- [24] G. Gogniat, M. Auguin, L. Bianco, and L. Pegatoquet, "Communication synthesis and HW/SW integration for embedded system design," in *Proceedings of the 6th International Workshop on Hardware/Software Codesign*, pp. 49–53, Seattle, Wash, USA, 1998.
- [25] A. Silva, G. Esmeraldo, E. Barros, and P. Viana, "Cache-analyzer: design space evaluation of configurable-caches in a single-pass," in *Proceedings of the International Workshop on Rapid System Prototyping*, pp. 3–9, May 2007.
- [26] MediaBench, 2006, <http://euler.slu.edu/~fritts/mediabench/>.
- [27] M. R. Guttaus, J. S. Ringenberg, D. Ernst et al., "Mibench: a free, commercially representative embedded benchmark suite," in *Proceedings of the 4th IEEE Annual Workshop on Workload Characterization*, pp. 3–14, December 2001.
- [28] J. Scott, L. Hwang Lee, J. Arends, and B. Moyer, "Designing the low-power M•CORE architecture," in *Proceedings of the International Symposium on Computer Architecture Power Driven Microarchitecture*, pp. 145–150, Barcelona, Spain, July 1998.