Cycle-Accurate Power Analysis for Multiprocessor Systems-on-a-Chip

Mirko Loghi
Dipartimento di Informatica
University of Verona
Strada Le Grazie, 15
37134 Verona, Italy
loghi@sci.univr.it

Massimo Poncino
Dipartimento di Informatica
University of Verona
Strada Le Grazie, 15
37134 Verona, Italy
massimo.poncino@univr.it

Luca Benini
DEIS
University of Bologna
Viale Risorgimento, 2
40134 Bologna, Italy
lbenini@deis.unibo.it

ABSTRACT

Developing energy-aware software for multiprocessor systems-on-chip (MPSoCs) is a difficult task, which requires the knowledge of the distribution of the power consumption among several heterogeneous devices (cores, memories, busses, etc.). In this work we analyze the power breakdowns of power consumption for a complete MPSoC platform, under several application workloads and operating conditions. We leverage a complete-system simulation platform with accurate power models for all key hardware modules. Our analysis shows that caches and system interconnect dominate in the power breakdown, pointing out how software locality is meaningful not only for performance but also for energy optimization.

Categories and Subject Descriptors: C.4 [Performance of Systems]: Measurement techniques

General Terms: Measurement Performance

Keywords: Low Power, Multiprocessor, System-on-Chip

1. INTRODUCTION

Technology scaling is the driving force for developing increasingly complex digital systems. Multiprocessor systems-on-chip (MPSoCs) are becoming widespread, not only in advanced applications. Some current consumer application, as for example some cellular handsets, require dual-processor (DSP and RISC processor) chips and future multimedia devices will require even more parallelism. Embedded and mobile multimedia systems have strict power budget [1] because they are often battery-powered. Furthermore, device reliability is strongly affected by power dissipation, because it depends on the average working temperature.
Power reduction requires then significant efforts in every phase of the design flow, from technology to software. Increasing the abstraction level when looking for energy efficiency allows better results, but, at the same time, it poses significant challenges in power estimation.

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

Developing energy-aware software for MPSoCs is a very difficult task and it is, nowadays, an open problem. The estimation of the power distribution and the development of power-efficient applications is a quite mature research area for uniprocessor environments. For these systems well known techniques exist, but there are a lot of features to take into account when moving toward multiprocessor platforms. The power, in fact, is consumed as by the processing elements, as well by the on-chip memory hierarchy and by the interconnection systems among them all and the energy spent depends on the internal state of them all.
In this work we use MPARM, a multiprocessor simulation platform, which is both cycle and signal accurate, for running realistic applications in order to obtain accurate functional behavior and power and performance analysis of the system. With respect to fixed execution traces or statistic traffic generators, our approach can take into account complex behaviors that cause relevant macroscopic effect [2].
Experimental results show that at least half of the power is consumed by the caches and that the interconnection power usage vary from 11 to 15%. Exploring several system configurations, it turns out that bigger caches can introduce a gain in performance as well as energy optimization, thanks to a more effective exploitation of software locality.
This paper is organized as follows. Section 2 surveys the state of the art on software level power estimation. Section 3 provides a description of the simulation platform used in this work its implementation, as well as its software architecture. Section 4 describes the benchmarks used for the exploration and presents the results. Finally, Section 5 draws some conclusions on the experiments.

2. RELATED WORK

Significant research effort has been devoted to architectural power estimation and power modeling in the recent past. Early work by Tiwari et al. [3] introduced the concept of *instruction-level power analysis*, a model which associates power consumption to instructions or instruction pairs. Model construction (also called characterization) is performed by extracting the data from low-level simulations of the target microprocessor, or by measurements on a real chip. The approach is *black box*. No assumptions are made on the target microprocessor architecture. This methodology has been extended in various ways: enhancements for DSPs [4] and VLIWs [5] have been proposed. After

model construction, the power consumed by a program running on the processor can be estimated by using a standard instruction-set simulator for extracting an instruction trace, and by summing up the costs for the instructions.
Better resolution and accuracy (at the price of increased execution time), are obtained with *microarchitectural power models* [6, 7], which assume the knowledge of the internal microarchitecture (e.g., execution units, register file, etc.). In this approach, power macromodels are obtained for the main internal units (from lower-level simulations), and their contributions are summed up during the run of a micro-architectural simulator executing the target program. Micro-architectural approaches are needed to accurately estimate the power consumed by advanced processors (e.g. superscalar with dynamic instruction scheduling), where multiple instructions can be issued at any given cycle and instruction commit is out-of-order.
Even though the power consumed by the processor is significant, power is consumed also in the memory system, system busses, peripherals. In most of real-life systems, the contribution of the CPU to the power budget, is not dominant especially when compared with the memory hierarchy. Thus, system-level power estimation must account for all system components. Several researchers [8, 9, 10, 11] proposed *full-system estimators*, that couple instruction set simulators with CPU, memory, bus and peripherals power models.
In principle, power estimation of a processor-based system can always be performed by running programs instruction by instruction on a full-system power estimator. In many cases, however, this approach is too slow for power optimization and design space exploration. Hence, several techniques have been proposed for characterizing software power consumption at a coarser level of granularity than a single instruction. Macro-modeling techniques have been proposed for sub-routine calls [12] within an application program, for operating system calls [13, 14, 15], and even for entire tasks [16, 17].
All above mentioned techniques have been applied in a single-processor setting. We are aware of only one recently published approach to multiprocessor systems power estimation [18]. In this work, the authors couple a complete-system simulator (VirtuTech's Simics [19]) with a microarchitectural simulator (Wattch [6]): instruction traces produced by Simics are fed to Wattch for power estimation. This approach focuses on processor power, and does not consider the power consumed by the other system components (interconnect fabric, external memory, peripherals). Furthermore, simulation is not cycle-accurate, especially for what concerns the effects of complex system interconnects and bus contention.

3. THE MPARM MULTIPROCESSOR PLATFORM

3.1 Hardware description

In this work we use MPARM, a SystemC multiprocessor simulation platform (Figure 1) consisting of (i) a configurable number of 32-bit ARM processors, (ii) their private memories, (iii) a shared memory, (iv) a hardware interrupt module, (v) a hardware semaphore module, and (vi) the interconnect.

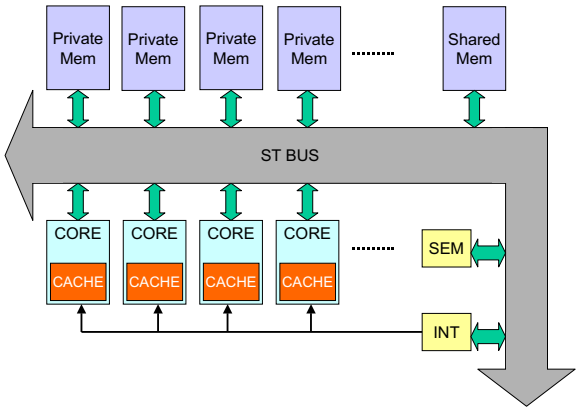

Figure 1: Hardware Architecture.

The processor cores are modeled by means of an adapted version of a GPL-licensed *Instruction-Set-simulator* (ISS) called SWARM [20], written in C++ and embedded into a SystemC wrapper. Each ISS contains his own cache, possibly split in instruction and data cache.
The memories and all the other devices are implemented in SystemC in a straightforward fashion. The semaphore module and the interrupt device are used to handle the synchronization among the cores. The former provides a *test-and-set* operation to the software, while the latter allows a processor to send an interrupt request to another core. Both these specialized hardware devices are mapped into the core's address space. To send an interrupt to a processor, an application writes a word into a specific location; to do busy waiting on a semaphore, it suffices to read from the proper address and loop until the operation return the correct value (namely, 0).
The platform is configurable, and it allows to specify several parameters:

- *The type of interconnect.* The 32-bit interconnection system can be an *AMBA AHB bus* [21] or a *ST-Bus* (a proprietary bus by STMicroelectronics). In addition, the topology of the ST-bus can be a *shared bus* configuration, a *full crossbar* configuration, or an intermediate *partial crossbar* topology. In this work, we focus on an ST-bus shared bus configuration.
- *The number of processing elements.*
- *The cache parameters.* In particular, we can specify the organization (split or unified), the type (direct-mapped, fully-associative or set-associative), the size and the line size.
- *The memory parameters.* In particular, we can specify the size and the latency. This applies also to the dedicated hardware (which is viewed as a memory); and the size of the memories.

The whole system is described in SystemC at signal level, except the ISS which performs cycle accurate simulations of the cores; the resulting simulations are signal- and cycle-accurate, respectively.

3.2 Software Implementation

Figure 2 shows the software architecture of MPARM. Names in parenthesis inside the boxes denote the names of the corresponding C++ class.

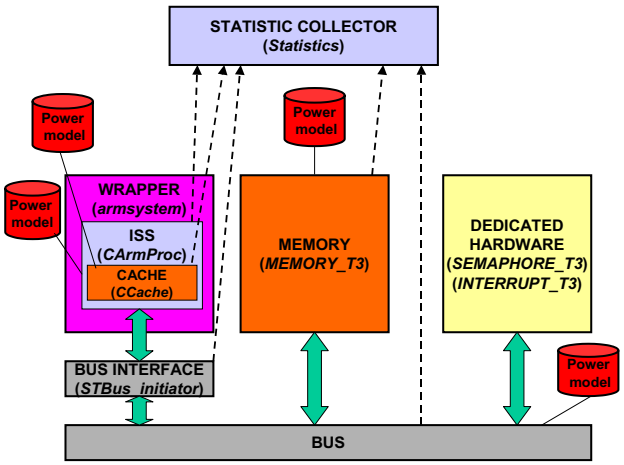

Figure 2: Software Implementation of the Platform.

The ISS is modeled as a C++ class (*CArmProc*) and is embedded into a SystemC wrapper (*armsystem*). The wrapper allows the usage of the ISS in a SystemC environ, while the bus interface (implemented in the class *STBus_initiator*) is in charge to translate the ARM requests toward the bus, to bus specific protocol requests.

In addition, the core contains its cache as a data member of type *CCache*. This is a base class from which other classes that implement specific cache types can be implemented: the fully associative (*CAssociativeCache*), the direct-mapped (*CDirectCache*) and set-associative cache (*CSetAssociativeCache*). The base class provides a common interface to access to the data from the code executed by the ISS.

The implementation of memories and of dedicated hardware is straightforward, due to their simple functional behavior. Memories are simply a class (*Memory_T3*) which embeds an array of data; the only potential difficulty lies in the interface toward the bus. The memory, in fact, must be able to understand the bus protocol and communicate accordingly. Dedicated hardware modules (*Semaphore_T3* or *Interrupt_T3*) can be thought of as a variant of memories. They also receive read or write requests and must act accordingly, depending on their specific operations.

The overall platform is instantiated in a dynamical way at the very beginning of the simulation in the `sc_main` method. Besides the classes which implements hardware devices, we developed some other class to perform data collection for performance and power consumption. The main one is the *Statistics* class which contains several methods to handle and record events. For instance, the `inspectMemoryAccess` and the `InspectCacheAccess` are called when a memory or a cache module is activated, respectively.

3.3 Power models

In order to obtain energy consumption data from the simulations, we have inserted in the platform power models for each component (the only exception being the semaphore and interrupt devices). Using the models, the platform can provide the energy spent by any of the different component on a cycle-by-cycle basis. The models are functions which compute the energy spent by the correspondent device, using information on the device's internal state.

For the cores we have used a state-based power model. We distinguish between a *RUNNING* state and an *IDLE* state, with distinct values of power consumption. An ARM core can be in *IDLE* state when its internal pipeline is stalled, for a data or instruction dependency, or when it is waiting for some data from the bus. In both cases the core is consuming a smaller (yet non-zero) amount of energy with respect to the *RUNNING* state. Power consumption figures have been obtained from an implementation of an ARM7 on a 0.13 μm technology by STMicroelectronics (0.055 mW/MHz for the *RUNNING* state, and 0.036 mW/MHz for the *IDLE* state).

For the memories (both caches and private memories) we have used an empirical model, derived from interpolation of data extracted from a memory generator by STMicroelectronics for the same 0.13 μm technology. The model is parameterized with respect to the memory size (in 32-bit words), and has been derived by least-mean square regression of a set of energy values obtained with the memory generator for different memory sizes. We explicitly distinguish between read and write accesses (for which there are two different power models).

The caches are regarded as a special type of memories consisting of two distinct cell arrays, the data and the tag memory. Given the fixed size of the memory word (32 bits), the cache parameters automatically define the size of the tag array(s), and the size of the data array(s). For instance, a 4KB unified, 2-way associative cache with 16-bytes lines will have two 24-bit tag arrays and two 128-bit arrays.

To accurately model cache power, cache accesses are decomposed into different access sub-types. For instance, writing a word in a cache consists of (i) a tag memory access and (ii) a data memory access. In addition, if the cache is n-way associative, only the bitlines corresponding to the desired way are activated. This is different, for instance, from the case of a cache refill, in which an entire line is written into the cache.

For this reason, we have defined a richer set of cache access modes, with different power models: *READDATA* and *WRITELINE* when a whole line of the data-memory is read or written, *READTAG* and *WRITETAG* when the tag field is read or written and *WRITEWORD* when a single word into a data memory line is written. Therefore, the power spent by a cache operation depends on the operation type, on the cache type (fully associative, direct-mapped or set-associative) and on the line size, for the last parameter affects the size of the tag-memory.

The power model for the ST-bus is relative to the same 0.13 μm technology, and has been provided by STMicroelectronics. It computes the power spent during a clock cycle using the number of *cells* which are in transit on the bus. This datum is available thanks to the signal-accuracy of the simulation, which allows to know how many devices are transmitting on the bus by simply analyzing the request and grant signals.

The mechanism used to invoke the various power models is described in Figure 3.

When a given module is activated the related power module function is invoked with the actual parameters carrying information on the device state. For the ST-bus power model this data is the number of cell in transit on the bus, while for memories and caches it is the memory size and the access

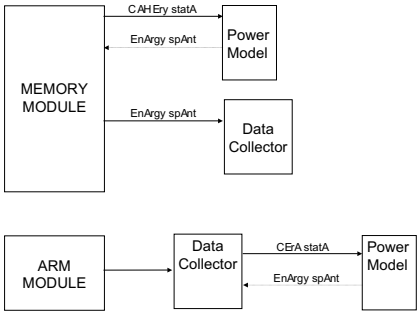

Figure 3: Invocation of Power Models.

type. The power module function returns to the caller (the module implementation) the amount of energy spent for the current operation. The module moves this value to the data collection routines which are in charge to gather and record information about performance and energy of the system. For the ARM core the information flow is different. The ISS, in fact, does not run when the core is stalled waiting for a bus response. Since the core is consuming energy also when idle, to collect this energy the power model is invoked from the data collector routine, which is activated at each cycle, and keeps track of the state of the core.

3.4 The Operating System

A port of a real-time operative system called RTEMS (Real-Time Executive for Multiprocessor Systems [22]) has been realized for this platform. User applications do link the RTEMS libraries to get access to the functionality of the embedded system, in the form of a set of system calls.

RTEMS is a lightweight and flexible OS for embedded systems, which it offers good support for multiprocessing, and provides native calls for communication and synchronization in such multiprocessor environments. Its targets are homogeneous and heterogeneous multiprocessing systems, for which it provides multitasking capabilities, flexible scheduling and a high degree of user configurability.

Inter-process and inter-thread communication in RTEMS rely on *message queues*. A thread which need to communicate with another thread must open a *message queue* which can be its own (local queue) or owned by another thread (remote queue). Both tasks involved in the communication must open the same queue, identified by a global ID, and one of them will own the queue. Then the transmitter thread uses the `rtems_message_queue_send` to put data into the queue, while the other uses `rtems_message_queue_receive` to retrieve them. The behavior of these two system primitives depends on the owner of the queue. It is different, in fact, to use a `rtems_message_queue_send` or a `rtems_message_queue_receive` on a remote queue with respect to a local queue. However, the differences in the queue handling are managed from RTEMS and are totally transparent to the user.

It is important to emphasize that inter-process communication primitives based on message passing impose precise restrictions on the programming model. Therefore, although the platform is built around a shared interconnect (which makes possible the explicit sharing of data), the programmer must stick to a message-based programming model. Although this may appear as a contradiction, it goes in fact already toward the support of different types of interconnect which scale much better with respect to the number of processors (e.g., crossbar topologies), but for which message-based communication is mandatory.

4. EXPERIMENTAL ANALYSIS

4.1 Description of the Benchmarks

As a test application for the evaluation of the system power consumption we have written an application which implements a digital filter. The filter is realized in two different configurations, resulting in two distinct applications, *FILT3-1* and *FILT5*.

The former, *FILT3-1*, works in a four-processor configuration. The first core performs an FFT of the signal, the second processor applies the filter to the signal and the third core executes the inverse FFT. The last processor executes the whole process (FFT, filtering, and inverse FFT) on different sets of data, implementing another filter which works in parallel with the first one (Figure 4).

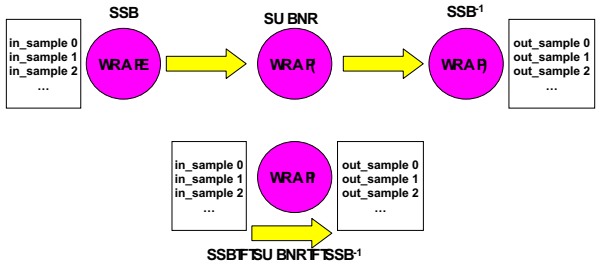

Figure 4: Data Flow for FILT3-1 application.

The second application *FILT5* is meant for a system with five processing units. The first two cores perform the FFT one half (Core 0 on the even-order samples, and Core 1 on the odd-order ones). The third processor (Core 2) applies the filter on all the samples. At the end the last two cores perform the inverse FFT on the two halves of the transformed data (Figure 5).

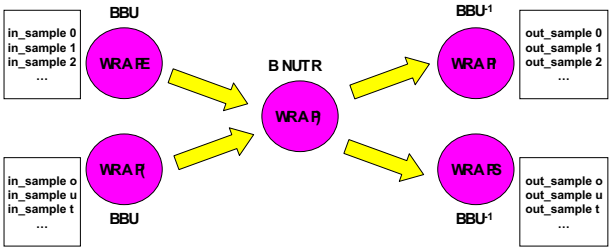

Figure 5: Data Flow for FILT5 application.

The data for both benchmarks are exchanged between processors using the RTEMS message-passing features. Depending on the size of the data, a single message could not be sufficient to carry all them. The application is thus in charge to split the data in more messages and to merge them later.

5. QUANTITATIVE ANALYSIS

In the first experiment, we ran the two applications using two different values of the number of signal samples (16 and

2048, respectively). Figure 6 and 7 show the power consumption breakdown for the *FILT3-1* and the *FILT5* application, respectively. We can notice that, in both cases, the main consumer of power are the caches. Their consumption range between 13 and 15% for *FILT3-1*, and between 10 and 14% for *FILT5*. The core power consumption is only about 5% of the total, while the power usage of the RAM is very low (around 1% for each private memory, and almost negligible for the shared memory). The other significant power consumption fraction is taken by buses, which consume a power fraction close to that of caches.

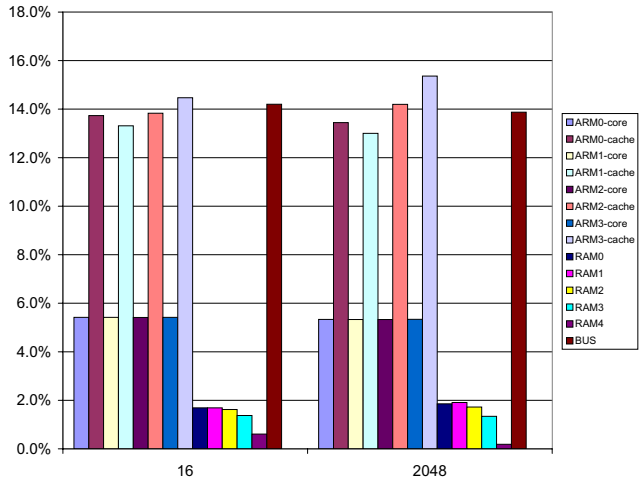

Figure 6: Power Breakdown for FILT3-1.

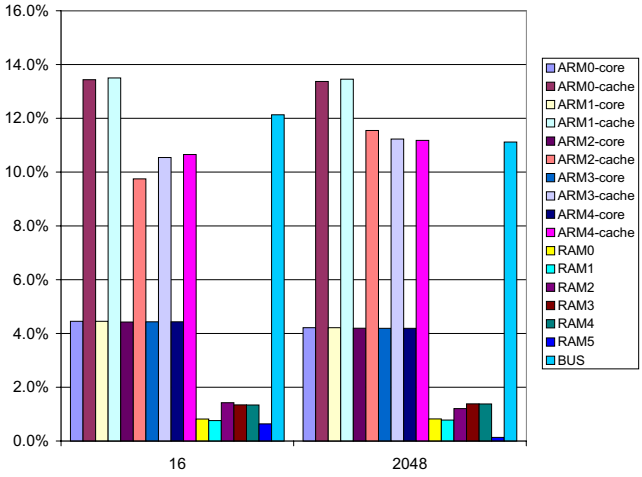

Figure 7: Power Breakdown for FILT5.

Focusing on the *FILT3-1* application, we notice that while all cores are consuming about the same level of energy, caches exhibit different consumption levels. ARM1-core, (the one involved in the execution of the filter) uses slightly less power than ARM0-core and ARM2-core, which perform the FFT and the inverse FFT, respectively. This trend is more emphasized for the case of 2048 samples, because of the nature of the application. ARM0-core and ARM2-core, in fact, tend to do more work on the same data, and use thus their caches more extensively. The shared memory power consumption uses the 0.61% of the power for the 16-sample case and the 0.19% for th 2048-sample one. This increase is due to the decreasing impact of the communication work with respect to the computation work, since the communication work grows linearly with the number of samples, while the computation work grows with $n \cdot log(n)$ (the complexity of the FFT algorithm).

A separate analysis deserves the ARM3-core (which does the entire work of transforming, filtering and inverse-transforming the data), and its private memory, RAM3. For the core the cache power consumption grows for increasing number of samples, while the memory uses a constant amount of power. Again, this is due to the increased cache-hit ratio. The cache-miss ratio decreases from about 4.7 every 1000 accesses, for the 16-sample case, to about 0.8 every 1000 accesses, for the 2048-sample case.

Some differences are observable in the *FILT5* application. In this case the workload is better balanced, because the processor which applies the filter works on data of double size, with respect to processors which performs the transform and the inverse transform. As for the other application, the core power consumption is about constant (about 4%) and the main sources of power dissipation are the caches and the bus, while the memories use very little energy. Again, the decreasing consumption of the shared memory (RAM5) when the sample size increases, indicates the major impact of computation with respect to communication.

In a second experiment, we analyzed the impact of different memory hierarchy configurations on the *FILT5* with 512 samples. In particular, we chose split instruction and data caches with three different cache sizes (1KB, 2KB, and 4KB for the data cache, and of double size for the instruction cache), as well as two different values of memory latency (1 cycles and 4 cycles). 1 cycle means that the memory must wait for a cycle before being ready to perform the access. We analyzed the two dimensions separately; Figure 8 shows the effect of different memory latency values, whereas Figure 9 shows the effect of different cache sizes.

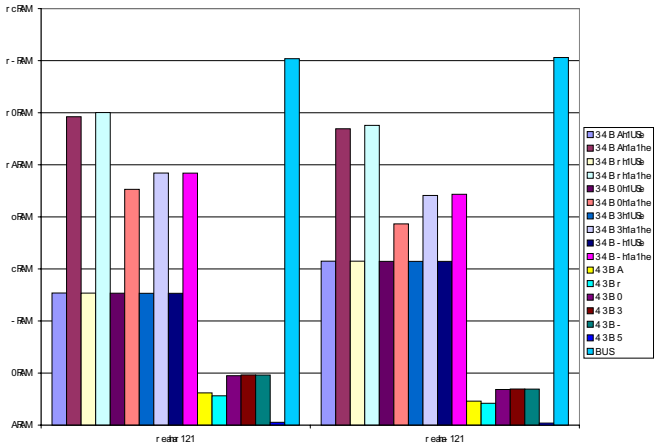

Figure 8: Power in FILT5 varying memory latency.

The best performance are obtained with the 4KB/1-cycle configuration, as expected, which is also the best solution for power consumption. This is due to the reduced number of bus and memory accesses, and to the low number of cycles in which the cores is stalled, waiting for a response from the memories.

Increasing the cache size seems to be a better solution than reducing the memory latency, both for performance and for energy consumption. This allows in fact to exploit program

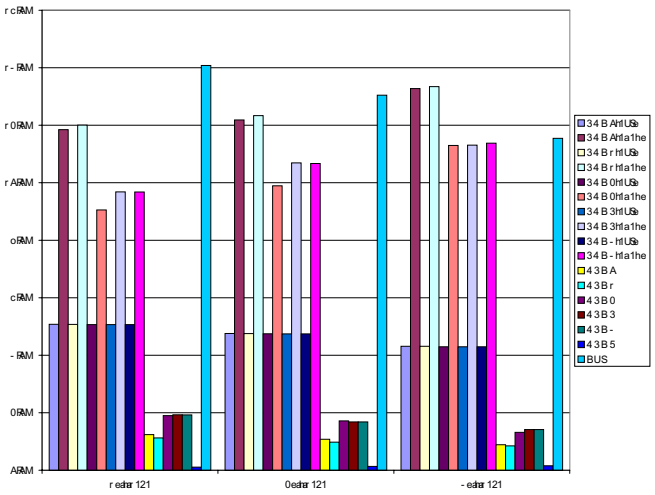

Figure 9: Power in FILT5 varying cache size.

locality and to reduce bus access. Reducing cache-misses is also an advantage, for it has an energy cost higher than a cache-hit. A cache-miss involves both a read and a write in cache, while for a cache-hit a read is sufficient.

Examining the power breakdown we can notice that increasing the memory latency increases the impact of core consumption; this is caused by the larger number of processor cycles when the processor is in the *IDLE* state, waiting for a response from the memory.

The power consumption share of the memories decreases when their latency increase. This is due to the fact that their energy consumption depends on the number of accesses, while the time spent for the accesses depends on the latency; therefore, it is intuitive that by increasing the access time, the energy should remain constant but the power should decrease.

Concerning caches, for larger sizes their power usage percentage increases, because the size has a direct impact on power consumption. This effect is limited from the increased performance and, therefore, from the reduced execution time.

6. CONCLUSIONS

Writing energy-aware applications in a MPSoC environment is a hard task: it requires the knowledge of the distribution of the energy consumption under several operating conditions, as well as the impact of the system parameters on the power usage. Using a cycle-accurate simulation framework it is possible to carry out a deep exploration of the power breakdown for such types of systems. This analysis can help driving programmers, but also system designers, in focusing their optimization onto the components that are the largest sources of energy consumption.

Leveraging a state-of-the-art multiprocessor simulation framework, in this work, we showed various power breakdowns resulting from different system configurations and for different applications.

7. REFERENCES

[1] L. Benini, M. Poncino, "Ambient Intelligence: A Computational Platform Perspective" in: *Ambient Intelligence: Impact on Embedded System Design*, T. Basten, M. Geilen, H. de Groot eds. Kluwer Academic Publishers, 2003.

[2] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini", "Analyzing On-chip communication in a MPSoC Environment", *DATE'04: Design Automation and Test in Europe*, to appear.

[3] V. Tiwari, S. Malik, A. Wolfe, "Power Analysis of Embedded Software: a First Step Towards Software Power Minimization," *IEEE Transactions on VLSI Systems*, Vol. 2, no. 4, pp. 437-445, Dec. 1994.

[4] R. Muresan, C. Gebotys, "Current Dynamics-Based Macro-Model for Power Simulation in a Complex VLIW DSP Processor," *IEE Proceedings - Computers and Digital Techniques*, Vol. 149, no. 4, pp. 173-187, July 2002.

[5] L. Benini et al. "A Framework for Modeling and Estimating the Energy Dissipation of VLIW-Based Embedded Systems", *Design Automation for Embedded Systems*, Vol.7, no.3, pp.183-203, Oct.2002.

[6] D. Brooks et al., "Power-Aware Micro-Architecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro*, Vol. 20, no. 6, pp. 24-44, Nov.-Dec. 2000.

[7] N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, W. Ye, D. Duarte, "Evaluating Integrated Hardware-Software Optimizations using a Unified Energy Estimation Framework," *IEEE Transactions on Computers*, Vol. 52, no. 1, pp. 59-76, Jan. 2003.

[8] T. Simunic, L. Benini, G. De Micheli, "Energy-Efficient Design of Battery-Powered Embedded Systems," *IEEE Transactions on Very Large-Scale Integration Systems*, Vol. 9, no. 1, pp. 15–28, Feb 2001.

[9] S. Gurumurthi, et al. "Using Complete Machine Simulation for Software Power Estimation: the SoftWatt Approach," *International Conference on High-Performance Computer Architecture*, pp. 124-133, 2002.

[10] J. Henkel, Y. Li, "Avalanche: an Environment for Design Space Exploration and Optimization of Low-Power Embedded Systems," *IEEE Transactions on VLSI Systems*, Vol. 10, no. 4, pp. 454-468, Aug. 2002

[11] T. Givargis, F. Vahid. J. Henkel, "Instruction-Based System-Level Power Evaluation of System-on-a-Chip Peripheral Cores," *IEEE Transactions on VLSI Systems*, Vol. 10, no. 6, pp. 856-863, Dec. 2002.

[12] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, "Cosimulation-Based Power Estimation for System-on-Chip Design," *IEEE Transactions on VLSI Systems*, Vol. 10, no. 3, pp. 253-266, June 2002.

[13] T. Tan, A. Raghunathan, N. Jha, "Embedded Operating System Energy Analysis and Macro-Modeling," *International Conference on Computer Design*, pp. 515-222, 2002.

[14] A. Acquaviva, L. Benini, A. Ricco', "Energy Characterization of Embedded Real-Time Operating Systems," in L. Benini, M. Kandemir, J. Ramanujam, *Compilers and Operating Systems for Low Power*, Kluwer Academic Publishers 2003.

[15] R. Dick, G. Lakshminarayana, A. Raghunathan, N. Jha, "Analysis of Power Dissipation in Embedded Systems using Real-Time Operating Systems," *IEEE Transactions on CAD*, Vol. 22, no. 5, pp. 615-627, May 2003.

[16] R. Dick, N. Jha, "MOGAC: a Multi-Objective Genetic Algorithm for Hardware-Software co-synthesis of distributed embedded systems," *IEEE Transactions on CAD*, Vol. 17, no. 10, pp. 920-935, Oct. 1998.

[17] A. Acquaviva, E. Lattanzi, A. Bogliolo, L. Benini, "A Simulation Model for Streaming Applications over a Power Manageable Wireless Link," *European Simulation and Modeling Conference*, Oct. 2003.

[18] J. Chen, M. Dubois, P. Stenstrom, "Integrating complete-system and user-level performance/power simulators: the SimWattch approach," *IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 1-10, 2003.

[19] P. Magnusson, et al., "Simics: a full system simulation platform," *IEEE Computer*, vol. 35, n. 2, pp. 50-58, Feb. 2002.

[20] Software ARM, `www.g141.com/projects/swarm`.

[21] ARM Ltd., `www.arm.com/products/solutions/AMBAHomePage.html`

[22] RTEMS home page, `www.rtems.com`.