A Memory Management Approach for Efficient Implementation of Multimedia Kernels on Programmable Architectures*

M. Dasigenis, N. Kroupis, A. Argyriou,
K. Tatas, D. Soudris, A. Thanailakis
VLSI Design and Testing Center,
ECE Dept., Democritus University of Thrace,
67 100 Xanthi, Greece

N. Zervas
VLSI Design Lab
ECE Dept., University of Patras,
26500, Patras, Greece

Abstract

A methodology for power optimization of the data memory hierarchy and instruction memory, is introduced. The impact of the methodology on a set of widely used multimedia application kernels, namely Full Search (FS), Hierarchical Search (HS), Parallel Hierarchical One Dimension Search (PHODS), and Three Step Logarithmic Search (3SLS), is demonstrated. We find the power optimal data memory hierarchy applying the appropriate data-use transformation, while the instruction power optimization is done using suitable cache memory. Using data-reuse transformations, performance optimizations techniques, and instruction-level transformations, we perform exhaustive exploration of all the possible alternatives to reach power efficient solutions. Concerning the embedded processor ARM, the experimental results prove the efficiency of the methodology in terms of power for all the multimedia kernels.

1. Introduction

In the past, the major concerns of the VLSI engineers were designing efficient circuits in terms of area and performance; power considerations were rarely dealt with. In recent years however, power dissipation has emerged as a significant design constraint along with these traditional factors, and designers have tried to find heuristic approaches for designing power and area efficient systems. Several factors have contributed to this attitude change. The most important factor was the remarkable success and growth of the class of personal computing devices (portable desktops, audio and video based multimedia products) and wireless communications systems (personal digital assistants and personal communicators) which demand high-speed computation and complex functionality with low power consumption, since power consumption affects the battery service life and weight, cooling requirements, packaging costs as well as the circuit reliability. There also exists a strong pressure for producers of high-end products to reduce their power consumption. For this reasons, power consumption has emerged as a very significant design constraint, that has to be tackled, especially in the high levels of design, where the most significant savings can be achieved [1].

Generally speaking, two possible implementations exist in order to meet the processing constraints: (i) that is the use of a dedicated hardware architecture, and (ii) a number of programmable cores. In particular custom hardware designs are area and power efficient but they lack of the flexibility, since it is possible to execute only one algorithm every time. On the other hand, the programmable cores are less efficient in terms of power consumption and chip area, but they are more versatile since they allow us to execute multiple algorithms in the same target architecture, and because design flaws can be easily found and corrected. Of course, as time-to-market requirements of electronic systems demand ever faster design cycles, an ever increasing number of systems are built around a programmable processor that implements an every increasing amount of functionality in firmware running on the processor. Only the most time-critical tasks need to be implemented in hardware.

In multimedia and other applications that make use of large multidimensional array type data structures, energy consumption related to memory transfer and storage dominates over the total system energy. Hence, memory optimization should have top priority, starting from system specification and moving down the design flow. As it was demonstrated in recent studies [2], the memory system is the main power consuming unit in multimedia systems, which is justified by the following reasons: (i) the data dominated nature of multimedia applications, as it was stated before, and (ii) the power consumed in accessing off-chip memories, which is significant more than normal arith-

*This work was supported by the project ED 501 PENED '99 funded by G.S.R.T of Greek Ministry of Development.

0-7695-1056-6/01 $10.00 © 2001 IEEE

metic or logical operations. Thus, it is necessary to perform hardware and software power optimization techniques, in order to design a power efficient system.

The problem of designing power and area efficient embedded systems, is rather new and thus, the bibliography is relatively small [2], [3], [4], [5]. Specifically, Catthour et. al. [2] proposed a systematic methodology for the reduction of data memory power consumption in custom architectures. Zervas et. al [3] presented another research work, which target single programmable processor-based systems. Recently, some novel power optimization techniques are presented in [6], stating for the first time the importance of the instruction memory power consumption on embedded systems. Except from the impact of the data memory, embedded systems are characterized by one additional critical component that has a significant part in the power budget, which is the instruction memory of the programmable memory that stores the algorithm to be executed. Some authors studied the impact of this [4], [5] presenting techniques of code placement in main memory to maximize instruction cache hit ratio [4], or partitioning on-chip memory into scratchpad memory and cache [5], but their targeted only on the I-cache and I-mem neglecting the D-mem power consumption.

The problem of designing power and area efficient systems, comprises of two distinct problems: (i) the data memory optimization using small on-chip memories, and (ii) the instruction memory optimization using appropriate caches. In this paper we perform an exhaustive exploration of data-reuse transformations, performance optimizations and power transformations, in terms of area, power and performance for multimedia applications executed on embedded cores. After we have found an optimal data memory hierarchy, we perform instruction power optimization by using a suitable cache memory. Experimental results, illustrate the efficiency of our proposed methodology.

2. Proposed Methodology

Our target architecture model consists of: (i) multiple processing cores, N, each of which has its individual on-chip instruction memory, (ii) instruction cache memory, and (iii) data memory hierarchy. This architecture is an extension of a previous architecture [6], but is extended to include not only the instruction memory but also it's corresponding cache. In this paper we analyze the effect of the Instruction cache (I-cache) in the reduction of the power instruction dissipation, for four multimedia kernels executed on the target architecure with one processing core ($N = 1$).

The flow of the methodology used is depicted in Fig. 1. The entry point is a C description of the multimedia kernel. This algorithm is then transformed using high level software optimization techniques, and is fed to the ARMulator,an ARM processor emulator . According to the used data-reuse transformations, we can calculate power and area of the data memory hierarchy[6]. On the other hand ARMulator can be used to extract a memory access trace which can be filtered to extract data address information, in order to use a cache simulator to extract cache statistics. We aim at the determination of the optimal data memory hierarchy for reducing power due to a number of off-chips transfers, and the optimal I-cache memory for reducing instruction memory power consumption. This problem, regarding the power consumption estimation, has been divided into two distinct and equal important tasks, that a designer has to account for: (i) the corresponding data memory hierarchy (D-Memory), and (ii) the corresponding instruction memory with the inserion or absence of an I-cache. The first task studied in [6] in detailed manner. Here, we are trying to optimize the whole memory subsystem of our targer architecture and not only the data-memory as we did in our previous work.

The methodology proposed here is based on the previous approach [6], but is extended to include the instruction memory. In this paper, the exhaustively exploration of both the instruction cache together with the data memory hierarchy is discussed. The motivation to take into consideration the I-mem was the results obtained from previous work, that revealed the dominant role of the I-Memory power in the total memory power budget. Of course, this is valid when programmable processor cores are assumed.

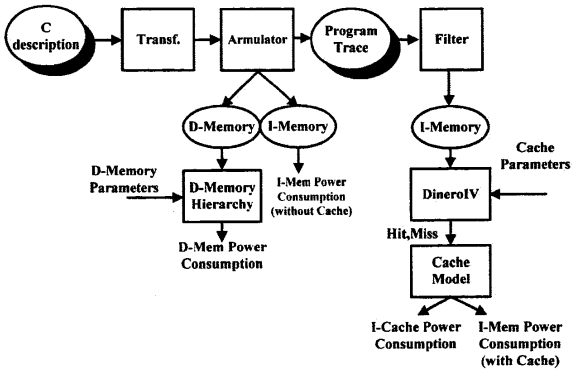

Figure 1. The proposed Methodology: Estimation of D-Mem, I-Mem, I-Cache power consumption

2.1. Data Memory Methodology

The first step of our data memory methodology is a set of software techniques aimed at optimization of the source algorithmic kernel in terms of power, performance and area, and consists of three types of high level transformations, namely data-reuse, performance and instruction level trans-

formations. The second step is to map the transformed algorithm to the physical memories, and analyze the efficiency of the methodology, with the experimental results. For the sake of completeness a brief analysis of the Data Memory Methodology follows, while detailed description can be found in [6].

Employing data reuse transformations [2], we determine the certain data sets, which are heavily re-used in a short period of time. The goals of these transformations are to reduce the redundancy in data transfers, and to introduce more locality in the accesses so that more data can be retained in memory cells in the memory hierarchy closed to the processing cores. The re-used data can be stored in smaller on-chip memories, which require less power per access. In this way, redundant accesses from large off-chip memories are transferred on chip, reducing power consumption related to data transfers. Energy for accessing the memory is reduced when memory banks are sufficiently small. On the other hand, a great number of different memories for each data set results into a significant area penalty, and imposes a severe wiring overhead, which tends to increase communication energy. For this reason, the data reuse exploration has to decide which data sets are appropriate to be placed in separate memory. Here, we applied 21 data-reuse transformations [3] to all target architecture models for the four ME kernels.

Another type of transformations applied was the performance optimizations, like common sub-expression elimination, loop optimization, tiling, interchanging, strip mining etc, that transform a loop for better temporal and spatial locality for a given cache size. Of course this kind of transformation has an impact in the instruction power budget. The tradeoff in this case was between the increase in the instructions due to the extra assignments in one hand, and the decrease in the instructions due to the use of the performance optimization. Sub-expressions are useful to eliminate when they have to be executed in a great number of loops. When the number of loops is small the overhead produced by the assignment retracts the benefits of the elimination.

The third type of transformations are the instruction level transformations, which are processor dependent. Every instruction dissipates a fixed amount of power, so if we are able to substitute instructions with high power consumption, with a set of instructions with lower total power consumption, we have a more power efficient algorithm. Indeed, a program written in high level language, e.g. C, can be re-written substituting power hungry instructions with power efficient ones. For example we have found that the multiply operation in the ARM processor could be substituted with summation operations.

The final step is the mapping process. For all the data-memory architectures models a shared background (probably off-chip) memory module is assumed. Thus, in all cases special care must be taken during the scheduling of accesses to this memory, to avoid violating data-dependencies and to keep the number of memory ports as small as possible in order to keep the power per access cost as small as possible.

2.2. Instruction Cache Methodology

One of the drawbacks of the programmable cores is the fact that the Instruction power consumption, has a significant part in the total power budget of the system, which is a motivation for the designer to try to decrease the I-mem power dissipation. Our approach to accomplish this task is by using one appropriate selected I-cache, which will be analyzed next.

After the data memory methodology has been applied to the original ME kernel, we have reached to some (in our case 21) transformed versions of our algorithm. These transformed algorithms imply the use of a specific data-memory hierarchy. Having this in mind we make measurements in order to evaluate the performance of the algorithms, in terms of area, power and performance, concerning the data memory hierarchy and the Instruction memory. The ARMulator environment can produce the exact memory access trace of the algorithm, in order to investigate the efficiency of a cache. The memory trace is then filtered to extract address information. Evaluation of cache statistics requires a cache simulator to account for the dynamic effects caused by the cache replacement mechanism, different degrees of associativity, write policy etc. For this reason, the filter transforms the output of the ARM into a format compatible with the Dinero cache simulator [7], and statistics are collected for various cache parameters. The statistics are used as input in the cache model in order to take measurements concerning power dissipation and chip area.

One of the disadvantages of the memory address trace of the ARMulator is the huge trace file that produces, which is not practical to use in order to extract the statistics using Dinero, which is executed in a different operating system too. For example the trace file of the PHODS algorithm, which has the smaller size comparing to the other three kernels, is 680 MB with over 20.000.000 instructions for the motion estimation of the limunance component of two consequence QCIF frames (144×176). This size includes the 45MB trace with instructions near 1.000.000 for reading the two frames. In order to analyze 21 transformations of the four ME kernels, the man's hours needed would be very inefficient. If we divide every frame into blocks 16×16 pixels, 99 blocks are formed, that is 9 blocks for the one dimension and 11 blocks for the other. The fact that these algorithms are block based means that the process of calculating the motion vectors is a recursive task for every block. Our belief was that we could analyze the trace for some blocks, and can expect that the statistics remain the same for the trace of all the blocks, with a low variation. Ex-

perimental results for all the algorithms justified this (Fig. 2 shows indicative results for the PHODS algorithm), and showed that the statistics for various cache parameters, for different number of blocks remain nearly unchanged, with the error fluctuation in the miss rate varies from 0.5% to 1%. For this reason we conducted our measurements for the I-cache for the derivation of motion estimation vectors of one block of the current frame only.

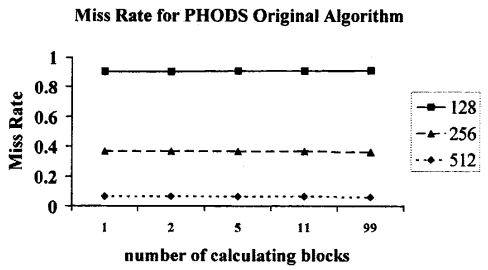

Figure 2. Cache statistics remain the same for different number of blocks

The combination of the data and instruction memory exploration/optimization provide a complete approach to the problem of excessive power consumption in an embedded system's memory. In order to take measurements for the I-cache power consumption we used an abstract cache model that is described in the next subsection.

2.3. Instruction Cache Model

Several cache models have appeared in the literature, although most of them are intended for performance, rather than energy estimation [8], [9]. All of these attempts to describe the power consumption of the cache are based on the circuit level, because the power dissipation is given as term of the capacitance, of the switching frequency of the address bus, the precharging energy etc. Although these models are accurate and very close to reality, they require a good knowledge of every specification of the target architecture, and the mathematic formulas are difficult to use for high-level power estimation.

In order to measure the power consumption of the I-cache we used an abstract model of the cache which depend only on the number of accesses in the cache. The number of accesses in the cache depends on the miss ratio m, as reported by the Dinero cache simulation, and the number of accesses n in the Instruction memory without using the I-cache, as reported by the ARMulator.

Specifying these two factors we can have an abstract estimation of the accesses in the cache and in the I-mem, which are:

$$I_{cache\ accesses} = (1+m) \times n \quad (1)$$

$$I_{mem\ accesses} = n \times m \quad (2)$$

The unity in Eq. 1, is justified by the fact that the processor core "requires" for the specific algorithm n accesses to the Instruction memory, whether exists or does not exist a cache. When there is a cache, all the n accesses will come from the cache. Of course there exist a miss rate of m, which means that are required $m \times n$ accesses into the Instruction memory, and in consequently $m \times n$ accesses into the cache memory. Summing these, we can calculate the I-cache accesses to $1 \times n + m \times n = (1+m) \times n$ and the I-mem accesses to $m \times n$. Using Landman's model, we can calculate the power consumption of the I-cache and I-mem, using the formulas of the power disipation of the RAM model.

The model we described above does not consider the energy dissipation due to the tag comparators, the steering logic (such as the internal decoder and the multiplexor), the cache control logic (used to implement the replacement policy), or the sense amplifiers, since they are regarded as minor sources of energy consumption. Therefore, it provides a lower bound of actual cache memory energy dissipation. For comparable sizes, cache accesses are intrinsically more energy-demanding due to the overhead required by the tag array and tag comparison logic.

3. Experimental results

In order to analyze the effectiveness of the combined I-mem and D-mem optimization, we used four well-known ME algorithms, namely: (i) Full Search, (ii) Hierarchical Search, (iii) Parallel Hierarchical One Dimensional Search, and (iv) Three Step logarithmic [10]. These algorithms are some of the fundamental multimedia cores that are in use in many multimedia systems on the consumer electronic market. Our experiments were carried out using the luminance components of QCIF frame (144x176) format. Reference window was selected to include 15x15 candidate blocks, while blocks of 16x16 pixels were considered. All these algorithms calculate the motion vectors of two images, but they differ in the granularity, the precision and the complexity. Specifically, FS is the most computational expensive but guarantees finding the optimal motion vectors, HS is a fast ME scheme that use a combination of search strategies that use both fewer search locations and fewer pixels in computing the motion vectors, while PHODS and 3SLS belong to the class of very fast algorithms that reduce motion-estimation complexity by reducing the number of search locations that are used in determining the motion vectors.

The key assumption in our methodology is that memory locations accessed by a multimedia application executed on

an embedded program are very localized, and for this reason the use of an I-cache can dramatically improve the power consumption of the I-mem, because accesses to a wisely selected I-cache are more cost-efficient than accesses to the I-mem. In order to verify this assumption, we have performed ME coding with the four kernels on two succesive frames, and we have collected statistics about data and instruction memory accesses. ARMulator, a software emulator for core processors of the ARM family, has been used as a memory profiler. ARMulator defines an external interface that allows us to provide models, written in C or C++, that can be simulated on the ARM platform.

Fig. 3 - 6 show the data memory and instruction memory (without cache) power consumption. Since the later factor is large, we use appropriate cache memories to achieve optimal instruction power consumption. Taking into account the code size of the innermost loop of each ME kernel, we perform measurements for cache memory sizes 128 bytes, 256 bytes and 512 bytes for all transformations. It is assumed that block line L=2 bytes and degree of associativity a=1. Table 1 provides the average power reduction/increase percentages for all kernels. The optimal solution for each kernel is shown in separate diagrams for all data-reuse transformations. Fig. 7 - 10 illustrate the measurements with maximal power reduction, for FS, HS, PHODS and 3SLS.

Algorithm	Cache Memory Size		
	128	256	512
FS	-21	**-59**	-41
HS	+7	-44	**-66**
PHODS	+20	-12	**-35**
3SLS	-2	**-81**	-37

Table 1. The impact of the cache memory on the total instruction power consumption (-: denotes power reduction, +:denotes power increase).

4. Conclusions

A complete methodology for designing power-efficient embedded systems for ME kernels, was presented. Application specific, data-memory hierarchy and instruction memory, as well as embedded programmable processing elements, were assumed. The proposed methodology had two goals: First, to design of an efficient data memory hierarchy, and second to use of a suitable instruction cache to reduce the heavy impact of the programmable instruction memory on total power consumption. The experimental results prove that an effective solution either in terms of power, can be acquired from the right combination of pro-

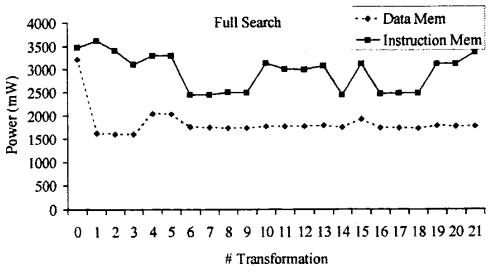

Figure 3. Data and Instruction power consumption in FS Algorithm

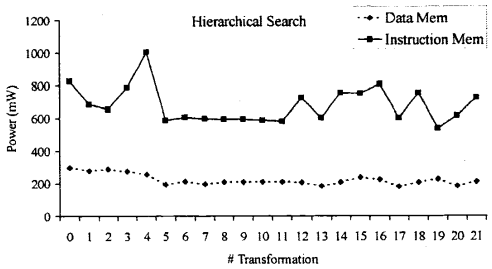

Figure 4. Data and Instruction power consumption in HS Algorithm

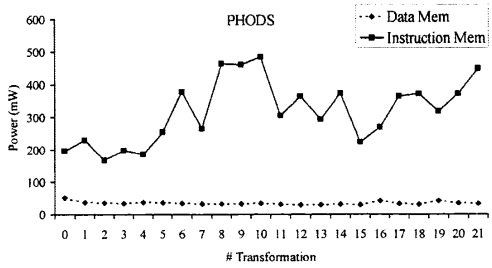

Figure 5. Data and Instruction power consumption in PHODS Algorithm

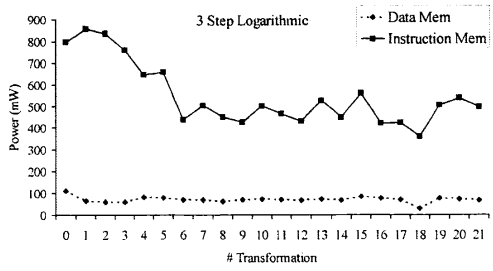

Figure 6. Data and Instruction power consumption in 3SLS Algorithm

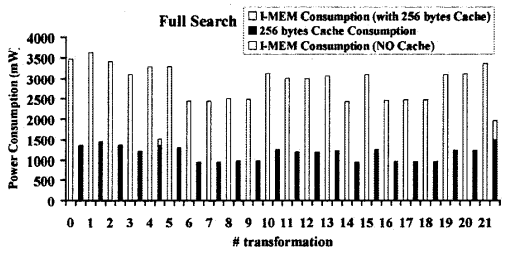

Figure 7. Cache Analysis for the Full Search Algorithm

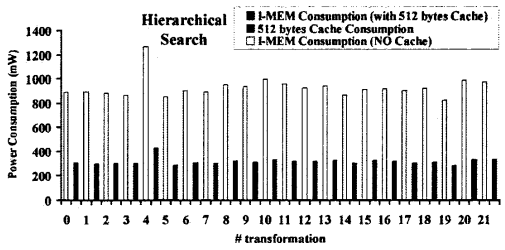

Figure 8. Cache Analysis for the Hierarchical Search Algorithm

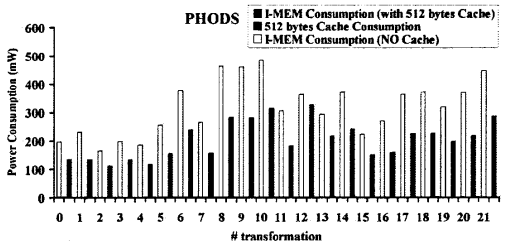

Figure 9. Cache Analysis for the PHODS Algorithm

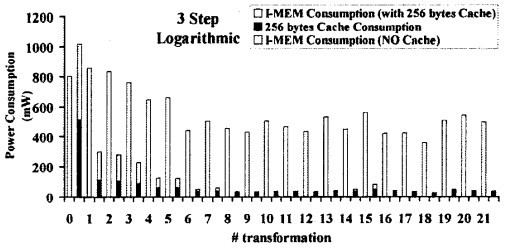

Figure 10. Cache Analysis for the 3SLS Algorithm

cessor core structure model, data-reuse transformation and suitable instruction cache.

Acknowledgments

The authors would like to thank Professors E. Macii and M. Poncino from the "Politechnico di Torino", for their help in DineroIV cache simulation application.

References

[1] A. P. Chandrakasan, R. W. Brodersen, *"Low Power Digital CMOS Design"*, Kluwer Academic Publishers 1998.

[2] F. Catthoor, et al., *"Custom Memory Management Methodology,"* Kluwer Academic Publishers, Boston, 1998.

[3] N. D. Zervas, K. Masselos, C.E. Goutis, *"Data-reuse exploration for low-power realization of multimedia applications on embedded cores"*, Proc. of PATMOS'99, October 1999, pp. 71-80.

[4] S. McFarling *"Program Optimization for Instruction Caches"*, Proc. of the 3rd Int. Conf. on Architectural Support for Programming Languages and Operating Systems, pp. 183-191, April 1989.

[5] P. R. Panda, N. Dutt and A. Nicolau *"Memory Issues in Embedded Systems-on-chip: Optimizations and Exploration"*, Kluwer Academic Publishers, 1999.

[6] D. Soudris, et. al., *"Data-Reuse and Parallel Embedded Architectures for Low-Power, Real-Time Multimedia Applications"*, Proc. of 10th Int. Workshop PATMOS 2000, Sep. 2000, pp. 243-254.

[7] Jan Edler and Mark D. Hill, *"A cache simulator for memory reference traces"*, http://www.neci.nj.nec.com/homepages/edler/d4

[8] M. Kamble and K. Ghose, *"Analytical Energy Dissipation Models for Low Power Caches"*, ISLPED-97 ACM/IEEE Int. Symp. Low Power Electronics and Design, Monterey, Calif., Aug. 19997, pp. 143-148.

[9] S. J. E Wilton, and N. P., Jouppi, *"An Enhanced Cache Access and Cycle TIme Model"*, IEEE J. Solid-State Circuits, vol. 31, no. 5, pp. 677-687, May 1996.

[10] V. Bhaskaran and K. Kostantinides, *"Image and Video Compression Standards"*, Kluwer Academic Publishers, 1998.