Evaluating the Effect of Data-Reuse Transformations on Processor Power Consumption

Alexander Chatzigeorgiou, Stamatiki Kougia, and Spiridon Nikolaidis

Electronics and Computers Division, Department of Physics,
Aristotle University of Thessaloniki, 54006 Thessaloniki, Greece
{achat, mkoug}@skiathos.physics.auth.gr,
snikolaid@physics.auth.gr

Abstract. Processor power savings that can be obtained by the application of data-reuse transformations on multimedia applications are discussed in this paper. Data Transfer and Storage Exploration methodologies primarily aim at memory related power reduction by moving data accesses to smaller memories, which are less power costly. However, it is shown that the applied code transformations have also a significant effect on the processor power consumption. Physical measurements of the average current drawn by instruction sequences provide a way to evaluate the processor power consumption of alternative code implementations. Simulation results prove that code transformations based on memory hierarchy exploration can have a significantly larger impact on power than existing software energy optimizing methodologies.

1 Introduction

Power consumption of embedded processors has been in the focus of several research efforts due to their widespread usage in portable systems, which are characterized by the intense requirement for low power. Sources of power consumption in such systems with varying importance according to the application, are data and instruction memory accesses, interconnect capacitance switching and instruction-level power consumption within the processor [1].

For data-dominated applications such as multimedia algorithms, the Data Transfer and Storage Exploration (DTSE) described by [2] as well as its extension for general-purpose platforms [3], [4] offer a complete methodology for obtaining and evaluating a set of data-reuse code transformations in terms of memory power. These transformations aim at moving data accesses from background memories to smaller foreground memory blocks, which are less power costly, resulting in significant power savings.

Interconnect related power consumption is due to switching of the large parasitic capacitances that are present in long interconnect lines. Several power reduction techniques have been developed for addressing this source of power consumption of which most important are appropriate data encoding schemes that minimize the average switching activity [5].

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

Concerning the internal processor power, extensive research on instruction-level power analysis has been performed by [6], [7], [8] based on physical current measurements for determining the average power consumed by a processor while running a set of instructions. Through these experiments, *base costs* as well as *inter-instuction overhead costs* have been associated with each instruction and classified according to different instruction categories. It should be mentioned, that processor energy consumption for a given program constitutes a significant portion of the total energy and consequently should be taken into account in the design of an embedded system. Moreover, design constraints such as required battery life, might impose a specific energy budget for the processor that has to be met.

Based on these measurements, software optimization techniques have been proposed, according to which the instructions with the lowest cost are selected to be scheduled (list scheduling algorithms). However, such methodologies are limited to instruction reordering as well as operand reordering of the same implementation without gaining advantage from any structural transformation of the code, which might result in lower power consumption.

In this paper, data-reuse code transformations are evaluated in terms of instruction related power consumption. Rearrangement of data between arrays can lead to significant simplification of addressing equations and therefore to drastic changes in the number of executed instructions. It will be proved that significant energy savings can be obtained by the exploration of all possible code transformations and the selection of the most suitable in terms of energy, taking into account the effect on memory related energy consumption. A number of conclusions drawn from the instruction level power analysis will be discussed in order to point out the significance of processor power exploration.

2 Data Reuse Transformations

In order to measure the effect of data-reuse transformations on processor power consumption the ARM 7 TDMI processor core has been considered since it is widely used in embedded applications due to its promising MIPS/mW performance. Moreover it offers the advantage of an open architecture to the designer [9].

Since the proposed study forms an integral part of a system-level power model, to enable the possibility of a flexible memory hierarchy that is suited to the applied code transformation, a data memory hierarchy consisting of several memory blocks communicating with the processor over a global bus is considered [2]. Data memories are considered to reside on chip except for the first memory layer that holds the previous and the current frame, which is an off-chip memory. The processor core is coupled with its own instruction memory, which is an on-chip single port ROM. Its size is determined by the code size, which in turn depends on the applied transformation to the original code.

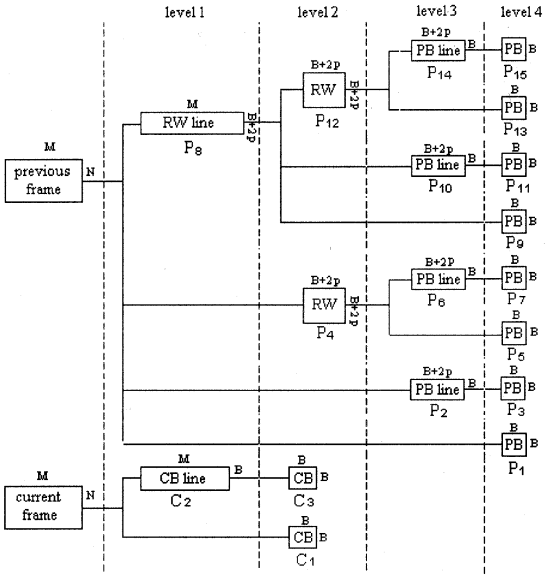


Fig. 1. Copy tree for a motion estimation kernel

To illustrate the power savings that can be gained by the application of data-reuse transformations, a typical motion estimation algorithm, namely the full search [10], will be used as test vehicle.

Cathoor et al [2] suggested that, based on the temporal locality of data memory references, a number of code transformations can be applied to any data intensive algorithm aiming at a memory hierarchy where copies of data from larger memories that exhibit high reuse are stored to additional layers of smaller memories. Since smaller memories have a lower energy cost per access, the total power consumption is reduced.

For motion estimation algorithms the possible data-reuse transformations together with the introduced levels in the memory hierarchy, which correspond to reused data sets, are shown in Fig. 1 [11]. The parameters for these algorithms are: the size of the current and previous frame ($N \times M$), block size ($B \acute{} B$) and p which determines the search region [$-p$, p] around the location of a specific block in the current frame. These transformations involve memories for a line of reference windows (RW line), a reference window (RW), a line of candidate blocks (PB line), a candidate block (PB), a line of current blocks (CB line) and a current block (CB). Each rectangle in the figure is annotated by the number of the corresponding transformation and the size of the introduced memory, given parametrically. Capital letters *C*, *P* indicate current and previous frame respectively, in which the transformation takes place.

As an example, one code transformation for the full search motion estimation algorithm where a line of reference windows is introduced is shown in Fig. 2. together with the original code (for reasons of clarity, all data in the introduced array are updated without exploiting the fact that some data already exist).

3 Instruction Level Power Models

According to the initial hypothesis by Tiwari et al [6] it is possible by measuring the current drawn by a processor as it repeatedly executes certain instructions, to obtain most of the information required to evaluate the power cost of a program for that processor. This claim has been refined to state that the total energy cost cannot be calculated by the summation of the energy costs of the individual instructions [6], [7], [8]. It has been proved that the change in circuit state between consecutive instructions has to be taken into account in order to establish accurate instruction level power models.

The two basic components of an instruction power model therefore are:

1. *Base Energy Costs*: These are the costs that are associated with the basic processing required to execute an instruction. This cost is evaluated by measuring the average current drawn in a loop with several instances of this instruction. Some indicative base costs for the most often instructions and their most usual addressing mode in the full search algorithm are shown in Table 1.

Original Code	**Transformed Code**
Introduction of a line buffer of reference windows for the previous frame (indicated bold)	

`for(x=0;x<N/B;x++)  //For all blocks`
`for(y=0;y<M/B;y++)  //in current frame`
 	
`for(i=-p;i<p+1;i++)  //For all candidate`
`for(j=-p;j<p+1;j++) //blocks`
 	
`for(k=0;k<B;k++)  //For all pixels`
`for(l=0;l<B;l++)  //in the block`
`{`
`read pixel in current frame;`
`if (current pixel displaced by i, j)`
`lies outside frame`
`previous pixel = 0;`
`else`
`read pixel from previous frame;`
`}`
	`for(x=0;x<N/B;x++) //For all blocks in a`
	`//line of blocks`
	
	`for(i=0;i<B+2p;i++) // For a line of`
	`for(j=0;j<M;j++)  // ref. windows`
	`{`
	`if (current pixel displaced by i)`
	`lies outside frame`
	`previous_line[i][j] = 0;`
	`else`
	`read previous_line from prev frame;`
	`}`
	`for(y=0;y<M/B;y++)`
	
	`for(i=-p;i<p+1;i++) //For all`
	`for(j=-p;j<p+1;j++)//candidate blocks`
	
	`for(k=0;k<B;k++)  //For all pixels`
	`for(l=0;l<B;l++) //in the block`
	`{`
	`read pixel in current frame;`
	`if (current pixel displaced by j)`
	`lies outside frame`
	`previous pixel = 0;`
	`else`
	`read pixel from **previous_line**;`
	`}`

Fig. 2. Original and Transformed code (transf. #4) for the full search algorithmic kernel

Type	Instruction	Addressing Mode	Base Cost (mA)
Arithmetic	ADD	LSL Immediate	9.92
	SUB	Immediate	6.67
	CMP	Immediate	6.65
	MOV	Immediate	8.07
Load/Store	LDR	Offset Immediate	10.76
	STR	Offset Immediate	8.55
Branch	B		8.73

Table 1. Base Costs for the ARM7 processor

	ADD	CMP	STR
SUB	1.24	0.13	2.42
MOV	1.35	1.10	2.64
LDR	3.29	2.77	0.80
B	1.25	1.03	2.00

Table 2. Overhead Costs (mA) for pairs of different instructions

2. *Overhead Costs*: These costs are due to the switching activity in the processor circuitry and the implied energy consumption overhead resulting from the execution of adjacent instructions. To measure the average current drawn in this case, sequences of alternating instructions are constructed. Some indicative overhead costs between pairs of instructions that are met in the full search algorithm are shown in the matrix of Table 2, for the addressing modes of Table 1. Overhead costs between instructions of the same kind are significantly smaller.

Therefore, the total energy consumed by a program executing on a processor can be obtained as the sum of the total base costs and the total overhead costs. Since the ARM processor has no cache and the execution of the full search algorithm does not lead to pipeline stalls, their effect on energy consumption has been ignored. Thus, energy is given by $E_p = \sum_i I_i \times V \times N_i \times t$, where I_i is the average current drawn by instruction #*i*, N_i the required number of clock cycles for instruction #*i*, V the supply voltage and t the clock period. For the results that will be shown next a supply voltage of 5 V and a clock speed of 20 MHz has been assumed.

4 Processor Power Evaluation Results and Discussion

In order to evaluate the effect of data-reuse transformations on processor power consumption all codes have been simulated using ARMulator in order to obtain the trace of executed assembly instructions. A simple *C* parser has been implemented to count all instruction occurrences as well as pairs of instructions and to assign a

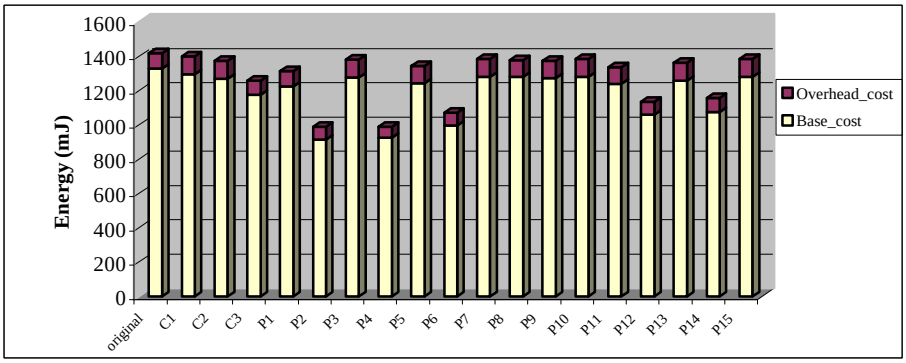


Fig. 3. Energy consumption for all data-reuse transformations

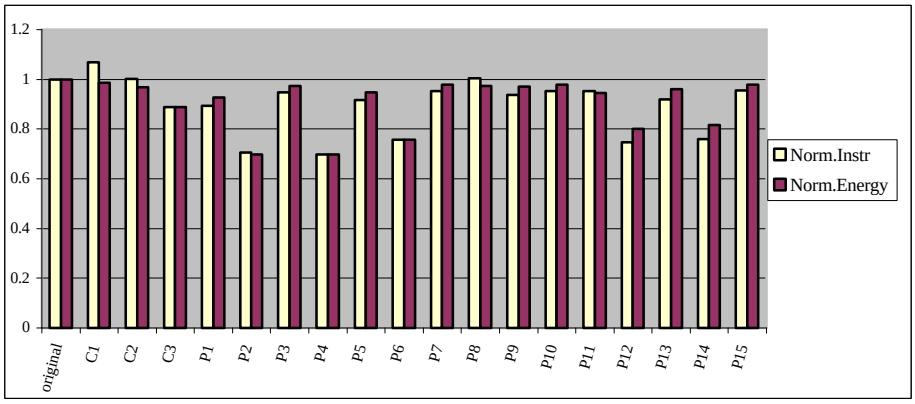
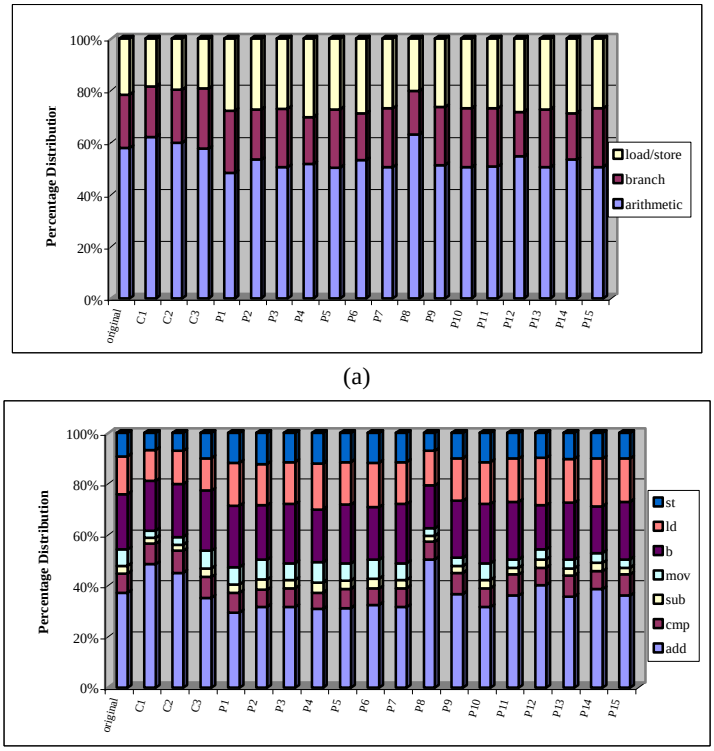


Fig. 4. Normalized instruction count vs normalized energy consumption

base/overhead cost using matrices containing the physical measurements provided by [8], [12]. To speed up instruction based power estimation, power can also be calculated for small segments of code determined by loop limits and then multiplied with the corresponding number of iterations.

The total energy, which is consumed during program execution is shown in Fig. 3 for all possible data-reuse transformations. In the same plot the contribution of base and overhead instruction cost to the total energy consumption of the processor is shown. In order to associate the relative effect of each transformation on power consumption to the number of executed instructions, the normalized instruction count is plotted versus the normalized energy in Fig. 4. In Fig. 5 (a) the distribution between main instruction categories is presented for all transformations while in Fig. 5 (b) the distribution is plotted considering the most often instructions in the algorithmic kernel.



(a)

(b)

Fig. 5. Instruction distribution between (a) main types and (b) most often instructions

From the presented simulation results the following conclusions can be drawn:
1. *Data-reuse transformations can have a significant effect on the number of executed instructions and the related energy consumption.* Since some transformations achieve energy savings of around 30%, a systematic power exploration employing instruction power models can yield significant optimizations, while according to [12] the application of scheduling algorithms to a set of different programs achieved maximum power savings of about 10%. For the case of the full-search motion estimation algorithm it is observed that in general, energy savings (for the processor) are better when no line of reference windows (RW_line) is placed in a separate memory block and when no memories corresponding to previous blocks (PB) are employed in the code. This is due to the fact that although the RW_line memory layer is accessed rarely, its size is quite large and on the other hand the PB memory is small but it is accessed very often.
2. *The amount of energy associated with overhead instruction cost is relatively small compared to base instruction energy costs* (overhead cost is between 6-8 % of the total cost). From this conclusion, it becomes obvious that software optimizing techniques (such as list scheduling) which aim to reduce the power consumption by rearranging instructions in order to achieve a lower overhead energy cost, are limited within the above range.

3. *The reduction in energy follows closely the reduction in instruction count,* due to the relative distribution of instructions, which remains unchanged between transformations (see conclusion 5). Thus, it is sufficient to explore the relative effect of each transformation on the number of executed instructions (provided by the processor simulator or an appropriate model [13]) in order to obtain the optimum solution in terms of processor power consumption. Moreover, performance in terms of instruction count, also determines the instruction memory power consumption for each transformation. It should be mentioned, that in other motion estimation algorithms (such as the three-step logarithmic search) the reduction in instruction count is even larger resulting according to the above observation in larger energy savings.
4. *Processor power reduction achieved by data-reuse transformations is primarily due to the simplification of the addressing equations in the most inner loops*. The introduction of additional memory layers, reduces the complexity of the addressing equations and the conditional statements in the most inner loops since data can be addressed using intermediate arrays in a more simpler manner and check of array boundaries can be performed using fewer inequalities. For example, considering the codes in Fig.2, in order to check in the original code whether the accessed pixel of the previous frame lies outside the frame, four inequalities have to be taken into account, implemented by 14 assembly instructions. For the transformed code, only two inequalities have to be checked leading to an implementation of only 7 instructions. Since these instructions are nested within 6 loops with a total number of 5.702.400 iterations, there is difference of seven times the number of iterations.
5. *The relative distribution of instructions between different types remains almost unaffected by the applied code transformation.* In other words, the effect of code transformations in terms of power consumption is due to the reduction of instruction count and not due to any rearrangement of instructions. Instruction reordering it known to have limited effect when overhead power costs are small compared to base instruction costs.
6. *Processor energy consumption is a significant portion of the total energy consumption of an embedded system.* Using results concerning data and instruction memory power for a different technology [13] and appropriate scaling, it follows that processor energy consumption is comparable to the energy that is consumed in the instruction and data memory during the execution of a given program. (It should be mentioned that instruction memory energy consumption is significantly greater than the energy consumed in the data memory, even in this data dominated application). Our work currently focuses on obtaining accurate results for the energy consumption of all system components, for the same technology, in order to evaluate which parts of the system benefit more from the application of data reuse transformations.

5 Conclusions

Data-reuse transformations, which primarily aim at memory related power reduction in multimedia algorithms have been evaluated in terms of processor power consumption. It has been shown that considering additional memory layers in the

algorithmic design, relaxes the complexity of addressing equations in the most inner loops, resulting in significant reduction in the number of executed instructions. As a result, power reduction around 30% compared to the original code has been achieved for some transformations. The main conclusion from the presented analysis is that energy consumption is proportional to instruction count since the application of data-reuse transformations does not alter the distribution of instructions. This exploration methodology, coupled with its direct effect on the number of memory accesses, can lead to large power savings in multimedia applications by determining the optimal implementation from a pool of possible solutions.

References

1. Chandrakasan A. and Brodersen R.: Low Power Digital CMOS Design. Kluwer Academic Publishers, Boston, (1995)
2. Catthoor F., Wuytack S., De Greef E., Balasa F., Nachtergaele L., Vandecappelle A.: Custom Memory Management Methodology. Kluwer Academic Publishers, Boston (1998)
3. Zervas N. D., Masselos K. and Goutis C.E.: Data-Reuse Exploration for Low-Power Realization of Multimedia Applications on Embedded Cores. Proc. of 9th Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), (1999) 71-80
4. Soudris D., Argyriou A., Dasygenis M., Tatas K., Thanailakis A., Zervas N.D., Goutis C.E.: Data-Reuse and Parallel Embedded Architectures for Low-Power, Real-Time Multimedia Applications. Proc. of 10th Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), (2000)
5. Benini L. and De Micheli G.: System-Level Power Optimization: Techniques and Tools. ACM Transactions on Design Automation of Electronic Systems, Vol. 5. (2000) 115-192
6. Tiwari V., Malik S. and Wolfe A.: Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. IEEE Transactions on VLSI Systems, Vol. 2. (1994) 437-445
7. Tiwari V., Malik S. and Wolfe A.: Instruction Level Power Analysis and Optimization of Software. Journal of VLSI Signal Processing. Kluwer Academic Press. (1996) 1-18
8. Sinevriotis G. and Stouraitis Th.: Power Analysis of the ARM 7 Embedded Microprocessor. Proc. of 9th Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), (1999) 261-270
9. ARM software development toolkit, v2.11, Copyright 1996-7, Advanced RISC Machines.
10. Bhaskaran V., Konstantinides K.: Image and Video Compression Standards: Algorithms and Architectures. 2nd edn. Kluwer Academic Publishers, Boston (1999)
11. Wuytack S., Diguet J.-P., Cathoor F.: Formalized Methodology for Data Reuse Exploration for Low-Power Hierarchical Memory Mappings, IEEE Transactions on VLSI Systems, Vol. 6. (1998) 529-537
12. Sinevriotis G. and Stouraitis Th.: SOFLOPO: Low Power Software Development for Embedded Applications. Public Final Report, European Commission, ESD Best Practice: Pilot Action for Low Power Design (2001)
13. Kougia S., Chatzigeorgiou A., Zervas N. and Nikolaidis S.: Analytical Exploration of Power Efficient Data-Reuse Transformations on Multimedia Applications. Proc. of International Conference on Acoustics, Speech, and Signal Processing (ICASSP), (2001)