IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'2001), May 7-11, Salt Lake City, Utah, USA.

ANALYTICAL EXPLORATION OF POWER EFFICIENT DATA-REUSE TRANSFORMATIONS ON MULTIMEDIA APPLICATIONS

S. Kougia, A. Chatzigeorgiou, N. Zervas[1], S. Nikolaidis

Section of Electronics and Computers, Department of Physics
Aristotle University of Thessaloniki, 54006 Thessaloniki, Greece
[1]VLSI Design Lab, Department of Electrical Engineering & Computers
University of Patras, Patras 26500, Greece

ABSTRACT

Power savings that can be achieved by data-reuse decisions targeting at a custom memory hierarchy for multimedia applications executing on embedded cores are examined in this paper. Exploiting the temporal locality of memory accesses in data-intensive applications a set of data-reuse transformations on a typical motion estimation algorithm is determined. The aim is to reduce data related power consumption by moving background memory accesses to smaller foreground memories, which are less power costly. The impact of these transformations on power, performance and area is evaluated both for application specific circuits and general purpose processors. The number of data and instruction memory accesses is analytically calculated, enabling a fast exploration of the design space by varying algorithmic parameters.

1. INTRODUCTION

Multimedia applications realized on embedded cores turn out to be data-dominated with the data-related power consumption affecting heavily the total power budget [1]. Real time applications such as image and video processing are increasingly being available on portable devices. Low power consumption is of primary importance for such systems since it determines their battery life and the maximum possible integration scale because of the related cooling and reliability issues [2].

A number of code transformations can be applied to any algorithm aiming at a memory hierarchy where copies of data from larger memories that exhibit high data-reuse are stored to additional layers of smaller memories. In this way, exploiting the temporal locality of data memory references, the greater part of the accesses is moved to smaller memories. Accesses to smaller levels of the memory hierarchy are less power costly and therefore significant power savings can be obtained [1].

Multimedia applications require increased performance and dedicated hardware in order to satisfy the requirement for high throughput of real-time programs. In order to confront this problem two implementation choices exist: The first is to use specific hardware (e.g. ASICs), which offers increased performance at a high cost. The second choice is to use embedded instruction set processors which offer increased flexibility and smaller time-to-market at the cost of lower performance than the previous solution.

In this paper a set of data-reuse transformations is examined using as demonstrator application the three-step logarithmic search motion estimation algorithm. A specific memory hierarchy is developed in order to exploit the presence of highly reused data sets in each transformation. The effect of each transformation on power, performance and area is evaluated for both application specific and general purpose platforms. For the first time, analytical expressions for the number of accesses to each data memory layer and to the instruction memory are extracted, enabling the fast exploration of the design space in order to determine the optimal solution.

2. DATA-REUSE TRANSFORMATIONS

In order to satisfy the requirements for high throughput and low power consumption of multimedia applications, an appropriate processor unit and data memory architecture has to be used. The target architecture is based on an embedded processor core with its own instruction memory, which is considered to be 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. The data memory hierarchy may consist of several memory blocks communicating with the processor over a global bus. Memory blocks are considered to reside on chip except for the first memory layer, which is an off-chip memory.

As test vehicle a typical motion estimation algorithm will be used: The two-dimensional logarithmic search which aims at reducing the computational complexity of the typical full-search algorithm by employing a heuristic search strategy for motion estimation similar to binary search. The algorithm structure is shown in Fig. 1, which has three double nested loops. For the calculation of the mean absolute error and the corresponding motion vector frames of size $N \times M$, blocks of size $B \times B$ and a reference window of size $(2p+B) \times (2p+B)$ are considered [3].

In the proposed approach only the power due to accesses to foreground and background memories is taken into account since the power due to accesses to register files is significantly smaller [4]. According to the power model that has been used, the power consumed on memory accesses is a function of the memory size, the access frequency, the technology, the number and the type (R or R/W) of ports and the number of bits per word.

In data-dominated applications such as multimedia algorithms significant power savings can be achieved by developing a custom memory organization that exploits the temporal locality in memory accesses. According to the

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

```
for(x=0;x<N/B;x++)        /* For all blocks in the current frame */
 for(y=0;y<M/B;y++)
 {  d=4;
    while(d>0)
    { for(i=-d;i<d+1;i+=d)        /* For all candidate blocks */
        for(j=-d;j<d+1;j+=d)
        {
          for(k=0;k<B;k++)        /* For all pixels in the block */
            for(l=0;l<B;l++) }
              check whether a pixel lies outside the frame
              read pixel in current and previous frame;
      d=d/2; } }
```

Fig. 1: Three step logarithmic search algorithm

proposed methodology data sets that are often being accessed in a short period of time are identified and placed into smaller levels of the memory hierarchy. Since smaller memory blocks have a lower energy cost per access, the total power consumption is reduced. Obviously, the total number of accesses to memory elements is increased since additional accesses are required in order to move data from the background to foreground memories.

The data-reuse exploration is performed by applying a number of code transformations to the original code, which are determined by the group of data sets that are being used in the algorithm. 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. 2. 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.

3. DESIGN EXPLORATION

As already mentioned there are two general approaches for the implementation of multimedia applications: Application specific IC's (non-programmable) or general purpose processors (programmable platforms). Since the power component due to instruction memory accesses is different for each approach, each case has to be studied separately:

3.1. Non-programmable platforms

In the case of an application specific integrated circuit the largest portion of the total power consumption is due to the accesses to the data memory. Application specific processors with a custom instruction set suited to the target algorithm are also considered to belong in this case. That is because the corresponding code size and number of executed program cycles is small. These in turn lead to a reduced memory size reducing significantly the power component due to instruction memory accesses. Consequently, the dominant power component is that due to data memory accesses.

In contrast to previous techniques [4] where each code transformation had to be evaluated by executing the corresponding code, it will be proved that it is not necessary to examine all possible transformations in order to evaluate their power efficiency. Rather, only a few key

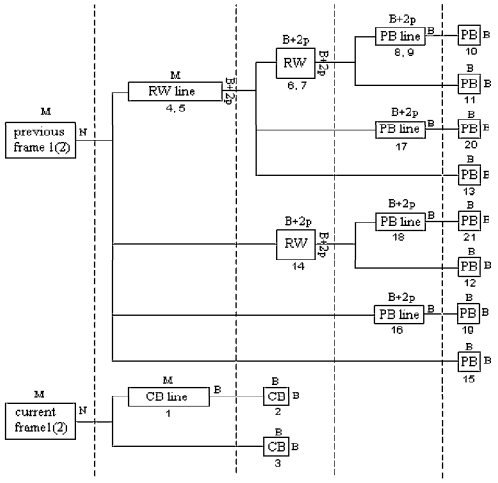

Fig. 2: Copy tree for a motion estimation kernel

transformations have to be studied in order to extract the required information.

The number of data accesses to each memory layer is the sum of the accesses, which are made in order to update this memory from its previous memory layer, and the accesses, which are made in order to update the next memory layer. It is obvious that the number of accesses, which are made in order to update a memory layer from a previous one, is independent of the previous layer from which data are read. Therefore, the number of accesses to a memory layer depends only on the following memory layer. According to the above, in order to calculate the number of accesses for each data transfer between memory layers, a table like Table I, has to be built. The contents of the cells provide the total number of accesses for the memory layer, which is indicated by the column, when it is followed by the memory layer that is indicated by the corresponding row. In order to calculate the total power consumption due to data accesses for each transformation, all involved memory layers have to be defined and for each memory layer its subsequent one has to be determined to find the entry in the Table that contains the corresponding number of accesses. For example, the accesses to data memory layers for transformation 10 are given in the shaded cells in Table I. For the case of the proposed memory hierarchy, all possible transformations are 21 while the required transformations in order to fill the table are only 8 for the previous frame and 1 for the current.

Another contribution of the proposed approach is that the total number of accesses to each memory layer is analytically calculated and the correspondent parametric expressions are shown in Table I. In this way the total number of accesses on each memory can be fed to the power model in order to evaluate the total power consumption. Consequently, the most power efficient solution can be determined very fast without having to execute each code on a simulator in order to count the number of accesses.

In Fig. 3 the total energy consumption due to accesses to data memory layers is presented for all transformations and compared to that corresponding to the original code. Since transformations on the previous and the current frame can be concurrently applied, two combinations of

accessed Layer / Next. Layer	0 (Previous)	1 (RW_line)	2 (RW)	3 PB_line	4 (PB)
0 (Previous)	$N \cdot M \cdot a^3$				
1 (RW_line)	$M \cdot (B+2 \cdot p) + (\frac{N}{B} - 1) \cdot M \cdot (B+1) - 2 \cdot p \cdot M$	$\frac{N}{B} \cdot M \cdot (B+2 \cdot p) + N \cdot M \cdot a^3 + (\frac{N}{B} - 1) \cdot M \cdot (2p+1)$			
2 (RW)	$\frac{N}{B} \cdot (B+2 \cdot p)^2 - \frac{N}{B} \cdot (B+2 \cdot p) \cdot p - 2 \cdot p \cdot (B+p) + 4 \frac{M}{B} \cdot (\frac{N}{B} - 1) \cdot B \cdot p$	$\frac{N}{B} M(B+2 \cdot p) + (\frac{N}{B} - 1) M(2p+1) + \frac{N}{B} (B+2 \cdot p)(B+1) + N(\frac{M}{B} - 1)(B+2 \cdot p)$	$\frac{N}{B} \cdot (B+2 \cdot p)^2 + \frac{N}{B} \cdot (\frac{M}{B} - 1) \cdot (B+2 \cdot p) \cdot (B+4 \cdot p) + N \cdot M \cdot a^3$		
3 PB_line	$\frac{N}{B} \frac{M}{B} \cdot (B+2 \cdot p) \cdot (3B+2p)$	$M \cdot (B+2 \cdot p) + (\frac{N}{B} - 1) \cdot M \cdot (B+1+4 \cdot p) + 3 \frac{N}{B} \frac{M}{B} (B+2 \cdot p)B + 2 \frac{N}{B} \frac{M}{B} (B+2p)p$	$\frac{N}{B} (B+2 \cdot p)^2 + \frac{N}{B} \cdot (\frac{M}{B} - 1)(B+2p) (B+4p) + \frac{N}{B} \frac{M}{B} (B+2p)(3B+2p)$	$\frac{N}{B} \frac{M}{B} (B+2p) + [3B+2((p-1)B-p)] + N \cdot M \cdot a^3$	
4 (PB)	$\frac{N}{B} \frac{M}{B} \cdot (9B^2 + 6Bp - 41) - 28N$	$M \cdot (B+2 \cdot p) + (\frac{N}{B} - 1) \cdot M \cdot (B+1+4 \cdot p) + \frac{N}{B} \frac{M}{B} \cdot (9B^2 + 6 \cdot B \cdot p - 2B - 18)$	$\frac{N}{B} (B+2 \cdot p)^2 + \frac{N}{B} (\frac{M}{B} - 1)(B+2p) \cdot (B+4 \cdot p) + \frac{N}{B} \frac{M}{B} (9 \cdot B^2 + 6 \cdot B \cdot p)$	$\frac{N}{B} \frac{M}{B} \cdot [24 \cdot B^2 + 34 \cdot B \cdot p - 4 \cdot p^2]$	$\frac{N}{B} \frac{M}{B} 3 \cdot B \cdot (15 \cdot B - 2p) + N \cdot M \cdot a^3$

Table I: Number of accesses to each memory layer according to the layer which follows ($a = \lceil \log_2 p \rceil$)

code transformations (7&3, 14&3) have also been examined. As expected, the power reduction becomes even larger when transformations on both frames are applied.

As it can be observed, the most power efficient transformation for the presented case (M×N=144×176, B=16, p=7) for the previous frame is transformation #14, while #3 is the best transformation for the current frame. One general remark that can be made is that for the current frame, transformation #3 yields always better results than the other two, since current blocks have no overlap and thus no advantage of a line of current blocks can be made.

Except for the fast calculation of the power consumption, these analytical expressions allow for the exploration of the whole design space by varying parameters such as the frame size (*N, M*), the size of the search space (*p*) and the block size (*B*). In Fig 4 the energy consumption for three code transformations is presented for varying frame sizes.

Since the introduction of additional memory layers comes with an area penalty, this parameter should also be taken into account. In Fig. 5 the effect of the proposed code transformations on area is illustrated. (Area is calculated using Mulder's model [5]).

3.2. Programmable platforms

Programmable platforms (general purpose processors) are obviously not optimally designed for each algorithm resulting in larger programs and therefore instruction memories and in higher number of executed cycles. Consequently, the power component due to instruction memory accesses is no longer negligible and has to be taken into account for the estimation of the total power consumption [4].

In order to prove the dominant role of instruction memory in the power consumption, simulations using the ARMulator have been performed [6]. In Fig. 6 the power consumption due to instruction memory accesses is shown as part of the total power consumption for the original and the transformed codes. As it can be observed, transformation #14 is no longer the most power efficient. It becomes clear that in the case of general purpose processors the number of accesses to the instruction memory as well as the instruction memory size should be efficiently evaluated in order to determine the best possible code transformation.

Similar to data accesses, the total number of executed instructions is calculated parametrically, according to the number and iterations of the nested loops that implement each of the applied motion estimation algorithms. In its general form, each double nested loop containing *m* instructions of the form :

```
for(i=0; i<n0; i++)
  for(j=0; j<n1; j++)
    { #m instructions }
```

corresponds to :

$$\#instr. = k_1 + k_2 \cdot n_0 + n_0 [k_1 + (k_2 + m) \cdot n_1] \quad (1)$$

assembly instructions. For the ARM processor k_1=4 and k_2=5. It should be noted that the number of *m* instructions within the loop, depends on the branch conditions imposed by the *if* statements for deciding whether a pixel in the reference window lies outside the previous frame or not. However, the number of times each of the logical criteria is fulfilled, is explicitly known from the previous analysis on data and consequently the exact number of assembly instructions can be obtained.

According to the proposed methodology, starting from the most inner loop, the number of executed assembly instructions is calculated and the result is added to the number of instructions between nested loops (which in turn

can be loops for introducing additional memory layers or single instructions). The final number of instructions is fed to the next outer loop until the total number of executed assembly instructions is obtained, resulting in a limited number of algebraic expressions. Since the indices of each loop are determined by the algorithmic parameters M, N, B and p, the total number of instructions is obtained as a polynomial function of these parameters. Consequently, the total number of accesses to the instruction memory, which is equal to the number of assembly instructions, can be efficiently evaluated leading to a very fast calculation of the instruction memory energy consumption.

It should be mentioned that in the results shown in Fig. 6, the power consumption due to instruction memory accesses is overestimated. That is because no instruction caching was taken into account, which for data dominated applications (where cache misses do not occur frequently) would result in a smaller number of accesses to the instruction memory.

Obviously, code transformations affect the processor performance, i.e. the number of cycles required for the execution of the code. In Fig. 7 the effect of the proposed code transformations on performance is illustrated.

4. CONCLUSIONS

A novel methodology for the evaluation of power efficient code transformations, which aim at the reduction of the data related power consumption for embedded processors implementing multimedia applications, has been presented. The transformations achieve power reduction by moving background memory accesses to smaller foreground memories. The effect of these transformations on power, performance and area has been examined for both general purpose and application specific platforms. Analytical expressions for the number of accesses to data and instruction memory are derived, allowing a fast exploration of the design space by varying all algorithmic parameters.

REFERENCES

[1] F. Cathoor, S. Wuytack et al., *Custom Memory Management Methodology*, Kluwer Academic Publishers, Boston 1998.

[2] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*, Kluwer, Boston MA, 1995.

[3] V. Bhaskaran, K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, 2nd ed., Kluwer Academic Publishers, Boston 1999.

[4] N. D. Zervas, K. Masselos and C.E. Goutis, "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'99), October 1999, pp. 71-80.

[5] J.M. Mulder, N.T. Quach, and M.J. Flynn, "An Area Model for On-Chip Memories and its Application", *IEEE Journal of Solid-State Circuits*, vol. SC 26, pp. 98-105, Feb. 1991.

[6] ARM software development toolkit, v2.11, Copyright 1996-7, Advanced RISC Machines.

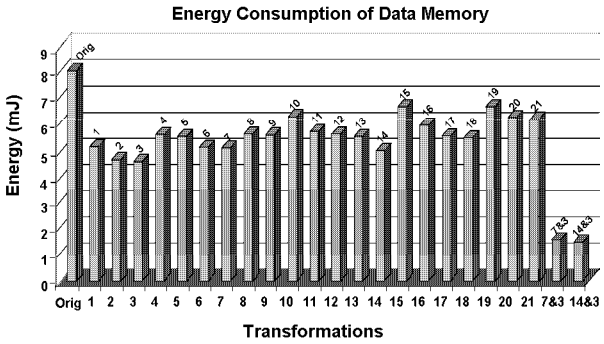

Fig. 3: Data Memory Energy Consumption

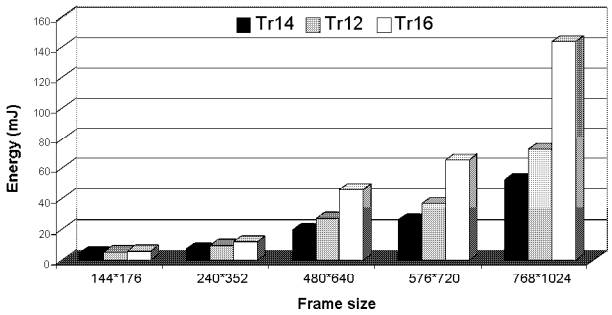

Fig. 4: Data memory energy consumption for three transformations / several frame sizes (B=16)

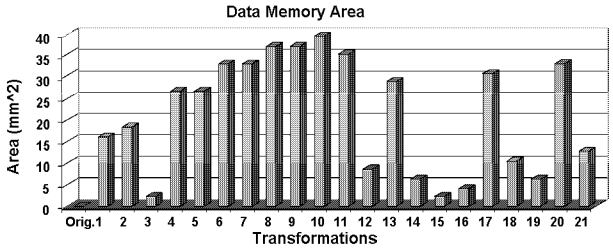

Fig. 5: Area occupied by data memory

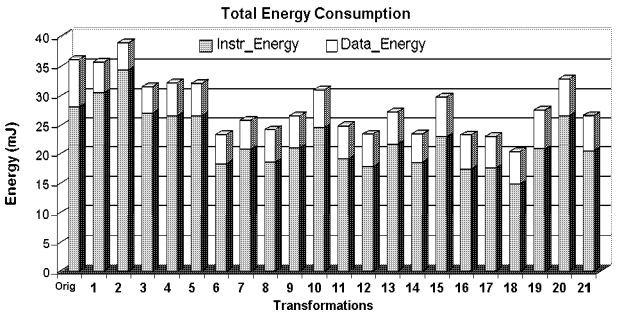

Fig. 6: Instruction memory energy consumption over total energy consumption

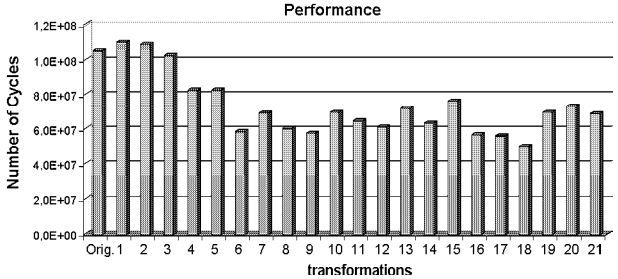

Fig.7: Code Performance for Different Transformations