Memory Hierarchy Optimization of Multimedia Applications on Programmable Embedded Cores[1]

K.Tatas, A. Argyriou, M. Dasigenis, and D. Soudris
VLSI Design and Testing Center, Dept. of Electrical and Computer Eng.
Democritus University of Thrace, 67100, Xanthi, Greece

N. Zervas
VLSI Lab Dept. of Electrical and Computer Eng.
University of Patras, 26500, Patras, Greece

Abstract

Data Memory hierarchy optimization and partitioning for a widely used multimedia application kernel known as the hierarchical motion estimation algorithm is undertaken, with the use of global loop and data-reuse transformations for three different embedded processor architecture models. Exhaustive exploration of the obtained results clarifies the effect of the transformations on power, area, and performance and also indicates a relation between the complexity of the application and the power savings obtained by this strategy. Furthermore, the significant contribution of the instruction memory, even after the application of performance optimizations to the total power budget becomes evident and a methodology is introduced in order to reduce this component.

1. Introduction

The number of multimedia systems used for exchanging information is rapidly increasing nowadays. The need for portable multimedia applications such as videophones, multimedia terminals and video cameras, as well as issues of cooling, packaging and reliability have made power consumption the predominant design consideration. Therefore, efficient power reduction techniques should be applied during the design of a system [1].

Additionally, these applications require increased processing power to manipulate large amounts of data, while meeting real-time constraints. This demand can be met by using either custom hardware architectures or a number of embedded programmable processors. The first option is power and area efficient, but it lacks flexibility since only a specific algorithm can be executed. On the other hand, the use of programmable processors may be less efficient for power and area, but it allows us to implement multiple algorithms on the same target platform and mainly meets time-to-market constraints easier. Either way, in order to meet real-time constraints, the initial application must be partitioned and assigned to a number of processing elements, in a power efficient way. For data dominated applications, like multimedia, the dominant factor in power consumption is the one related to data storage and transfers in custom-processor platforms [2]. In programmable platforms, on the other hand, the power required for instruction fetching, forces the role of the data storage and transfer power to diminish [3], [4].

In [4], partitioning of the Full-Search algorithm combined with data reuse transformations led to a reduction of the memory-related power cost. The transformations were applied after the partitioning step and either a power or power/delay or power/delay/area efficient combination of data reuse transformations and programmable processor architecture proved to be feasible.

Complex multimedia applications require the application of global loop transformations in order to be partitioned. Global loop transformations also make the subsequent application of data-reuse transformations significantly easier. In this paper that exact approach is followed and explored, while focusing particularly on a "Full-Search like" algorithm, namely Hierarchical Search Motion Estimation [5]. Memory hierarchy exploration is performed considering three multiprocessor

[1] This work was supported by the project PENED 99 funded by G.S.R.T of Greek Ministry of Development

0-7695-1025-6/01 $10.00 © 2001 IEEE

architectures. Performance optimizations [6] have also been applied and the on-chip/off-chip memory fetching latency has been explored in order to obtain more accurate measures on performance.

The results clearly indicate a direct relation between the complexity of the algorithm, i.e. number of operations, and the power savings obtained by this strategy and that the transformations mostly affect our target architectures in a similar way. Also, the dominant role of the instruction memory power to the total power consumption is dealt with by the use of suitable cache memory.

2. Target architectures

Each processor in the three considered multiple processor architectures has its own single on-chip instruction memory, the size of which is strongly dependent of the code size executed by the processor, and therefore it may vary from one processor to the other. We have dubbed this scheme application specific instruction memory (ASIM). Regarding the data memory organization, we assumed application specific data memory hierarchy (ASDMH) [2][7]. ASDMH is explored in combination with three well-established data-memory architecture models namely: 1) *distributed data-memory architecture (DMA)*, 2) *shared data-memory architecture (SMA)* and 3) *shared-distributed data-memory architecture (SDMA)*. For all data-memory architectures a shared background memory module (probably off-chip) is assumed. In the case of DMA a separate data memory hierarchy exists for each processor (Fig. 1). Therefore, all memory modules are single ported, but in the case of a lot of common data shared by the processors, a significant area penalty is probable. SMA implies a common memory hierarchy shared by all processors (Fig. 2). Since in the data-dominated programmable processing domain it is very difficult and performance inefficient to schedule all

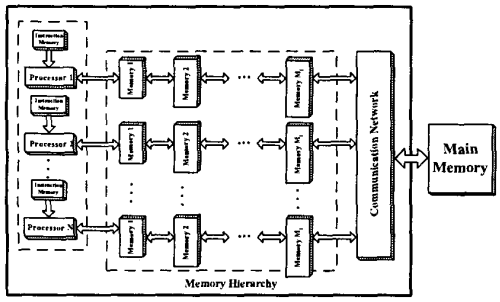

Figure 1. *The distributed data-memory architecture model*

memory accesses sequentially, we assume that the number of ports for each memory block equals the maximum number of parallel accesses to it. Finally, in the case of SDMA, being a combination of the above two models, the common data to the N processors are placed in a shared memory hierarchy, while a separate data memory hierarchy exists for the lowest levels of the hierarchy (Fig. 3). For demonstration purposes, we have examined the case of N =2 without any restriction about memory hierarchy levels.

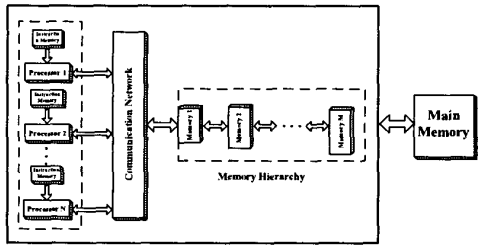

Figure 2. *The shared data-memory architecture model*

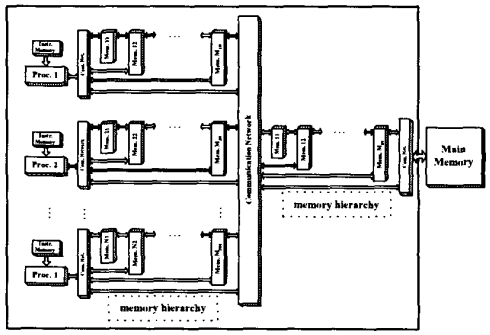

Figure 3. *The shared-distributed data-memory architecture model*

3. Global loop and data reuse transformations and performance optimizations.

Our goals are i) memory hierarchy exploration and optimization and ii) partitioning of the algorithm. Concerning the first objective there is an existing methodology known as Data Storage and Transfer Exploration (DTSE) [1], which dictates the following order of steps: i) Application of global loop transformations, ii) application of data-flow transformations iii) application of data-reuse transformations. In the specific application tackled here, direct application of data-reuse transformations would be difficult and it would not lead to optimal power savings, due to the nature of

the algorithm. The application of global loop transformations though, not only makes the subsequent application of data-reuse transformations significantly easier, but it also enables the partitioning scheme employed here, known as LSGP [7] allowing us to implement our application in a multiprocessor environment.

An alternative partitioning scheme would have been task-level partitioning, which would not require applying global loop transformations. In other words, we could have assigned to each processor a different task (or number of tasks) for the entire range of data, but then the problem of synchronization between the different processors would have occurred.

The use of manipulation techniques of memory data transfers has been explored and applied to motion estimation algorithms before [1],[3],[4]. The basic concept is to store data that is heavily reused in a short amount of time in smaller on-chip memories, which require less power per access. The key is to decide which data sets are appropriate to be placed in a (small) separate memory. Otherwise, a number of different memories will be required for each data set resulting to significant area penalty.

Furthermore, a category of performance optimizations has been used, known as common subexpression elimination [6]. They were applied in order to reduce the consumed instruction memory power and consequently the total consumed power. Performance is also affected positively.

We obtained our results using the same procedure as in [4]. The on-chip power is considered to be equal to the power related to the memory access itself by ignoring the power related to the on-chip bus activity because it is very small. For calculating these, Landman's memory model is used [9]. In order to estimate the performance of a particular application, we use the number of executed cycles resulting from the considered processor core simulation environment. Here, for experimental reasons we will use the ARMulator [10]. For the area occupied by the memories, Mulder's model is used [11].

We originally employed data-reuse transformations to reduce the consumed data memory power. In order to do the same for the instruction memory power we have introduced instruction caches. The exact instruction trace of the executed algorithms has been fed to the DineroIV cache simulator [12] for various cache parameters. The transformation with the biggest hit ratio and as small memory size as possible has been selected. Therefore, a cache that reduces the instruction fetching power has been selected.

4. Experimental results – comparative study

In this section, results are shown and comparative study of the relation between data-reuse transformations and data-memory models, assuming the application's partitioning. Firstly, we give a description of the algorithm and of the partitioning scheme used, and then the experimental results (power, area and performance) for all architecture models are presented and discussed.

4.1 Demonstrator application and partitioning

As mentioned before, for demonstration purposes we selected a widely used motion estimation algorithm, namely the 3-level Hierarchical Motion Estimation algorithm [6]. It is a fast motion estimation scheme that uses fewer search locations and fewer pixels in computing the mean absolute error (MAE) value. Our experiments were carried out using the luminance components of QCIF frame (144×176) format. The reference window was selected to be 15×15 candidate blocks, while blocks of 16×16 pixels were considered. Two low-resolution versions of the current picture and the reference picture are formed by subsampling both of them by a factor of two and four. The motion vector search begins at the lowest resolution picture (level 2). The search space is one fourth of the original one. At level one the motion vector search is performed with the origin being the block that corresponds to the level 2 block where the MAE is minimized and a search region of [-1,1] pixel around it. At level zero the search origin is located at the block which corresponds to the level one block where the MAE is minimized and the search region is again [-1,1] pixels around the block. The location that yields the smallest MAE corresponds to the final motion vector output. A Full-Search method is used for motion estimation at each level of the hierarchy, therefore the algorithm structure is very similar and is presented in pseudo-code in Fig 4. A more detailed description of the algorithm can be found in [6].

Partitioning was done using the LSGP technique [7]. In the case of p partitions the form of the partitioned algorithm is the one shown in Fig. 5. Our experiments were carried out assuming p=2, meaning 2 partitions. Therefore the first processor

executes the algorithm for loop index x range 0 to 4 and the second one for 5 to 8, in a parallel fashion.

```
subsampling_by_2();
subsampling
_by_4();
\* in subsampled by 4picture*\
for (x=0;x < N/B;x++)
for(y=0;y<M/B;y++)
        for(i=-p;i<p+1;i++)
        for(j=-p;j<p+1;j++)
                for(k=0;k<B;k++)
                for(l=0;l<B;l++)

        if((B*x+i+k)<0||(B*x+i+k)>N-
1||(B*y+j+1)<0||(B*y+j+1)>M-1)
                        \*conditional statement for the
pixel of canditate block *\
\* in subsampled by 2 picture *\
for (x=0;x < N/B;x++)
for(y=0;y<M/B;y++)
        for(i=-1;i<1+1;i++)
        for(j=-1;j<1+1;j++)
                for(k=0;k<B;k++)
                for(l=0;l<B;l++)

        if((B*x+i+k)<0||(B*x+i+k)>N-
1||(B*y+j+1)<0||(B*y+j+1)>M-1)
                        \*conditional statement for the
pixel of canditate block *\
\* in original picture *\
for (x=0;x < N/B;x++)
for(y=0;y<M/B;y++)
        for(i=-1;i<1+1;i++)
        for(j=-1;j<1+1;j++)
                for(k=0;k<B;k++)
                for(l=0;l<B;l++)

        if((B*x+i+k)<0||(B*x+i+k)>N-
1||(B*y+j+1)<0||(B*y+j+1)>M-1)
                        \*conditional statement for the
pixel of canditate block *\
```

Figure 4. *The Hierarchical Search motion estimation algorithm*

```
Do in parallel:
Begin
for(x=0;x<⌈N/(pB)⌉;x++){sub-algorithm}
for(x=⌈N/(pB)⌉;x<⌈2N/(pB)⌉;x++){sub-algorithm}
:
for(x=⌈((p-1)N)/pB⌉;x<⌈N/B⌉;x++){sub-algorithm}
end;
```

Figure 5. *The partitioned Hierarchical Search algorithm*

A comparative study of this partitioning scheme in combination with 21 data reuse transformations on power, performance and area was undertaken. The above mentioned memory hierarchies were applied to level 0, where the maximum power savings can be obtained since the memories are big enough. At levels 1 and 2 (subsampled frames) the memories are too small to justify the insertion of buffers. Our exploration showed that even applying data reuse transformations to all levels, does not give significantly better results than the ones obtained by applying the transformations only to the highest level.

We examined the effect of the target architectures described in the previous section in combination with 21 data reuse transformations [3],[4] on power, performance, and area. The transformations were applied after partitioning. They involved the insertion of memories for a line of current blocks (CB line), a current block (CB), a line of candidate blocks (PB line), a candidate block (PB), a line of reference windows (RW line), and a reference window (RW). In Fig. 9, the copy tree of the motion estimation algorithm is shown. It is identical for all processors and target architectures and the dashed lines indicate the memory hierarchy levels. Each rectangle contains three labels, where the number implies the applied data reuse transformations associated with the memory hierarchy level. The remaining two labels determine the size of the PB and CB line or block, RW line or reference window. In the case of the hierarchical motion estimation algorithm, there should be additionally four rectangles on the left side corresponding to the subsampled frames, but since we applied no data reuse transformations on them, due to the optimization mentioned above, the copy tree degenerates to the one in Fig. 6.

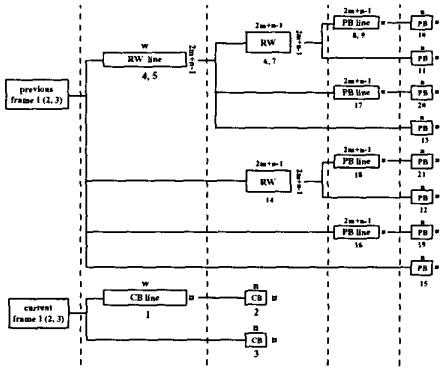

Figure 6. *The copy tree structure of the hierarchical search motion estimation algorithm*

4.2 Experimental results

Comparisons among the three target architectures and in terms of power, performance and area are shown in Fig. 7, 8, 9 and 10. A comparison concerning data power consumption for all models between Full Search and Hierarchical Search is illustrated in Fig. 11. Fig. 12 illustrates the results of our instruction power

estimations with and without instruction memory cache for only the transformations corresponding to the minimum and maximum instruction power consumption.

Fig. 7 illustrates the effect of the data-reuse transformations on the consumed data power for all models. The DMA (with one background memory) seems to be the least efficient, while the other two models are almost equally efficient. The effect of the transformations though, seems to be similar for all three models.

The effect of the data-reuse transformations on total power is depicted in Fig. 8. The SMA seems to be the most efficient of the three models. A comparison with Fig. 7 leads to the conclusion that the contribution of the instruction power to the total consumed power is significant. Often the data memory savings are overrun by the high instruction memory power consumption as is the case between the DMA and the SDMA.

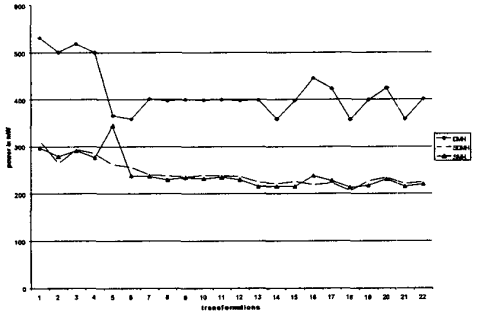

Figure 7. *The effect of data-reuse transformations on data memory power*

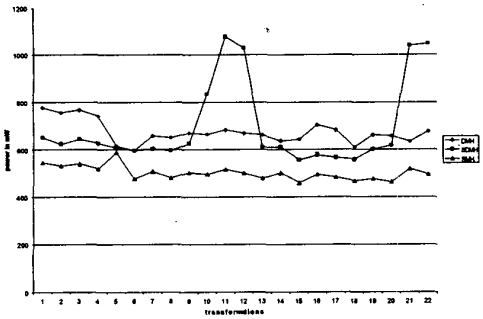

Figure 8. *Comparison results for total power*

The effect on performance is shown in Fig. 9 and is also similar for all three architectures. The SDMA is clearly the least efficient due to the increased code size related to control operations specifying which memory modules should be accessed. The other two architectures are equally efficient.

Fig. 10 illustrates the effect of the transformations in the total occupied total memory area. It can be seen that the transformations increase the area in comparison to the original, because they impose the addition of extra memory hierarchy levels. However, each transformation affects all models in an identical manner. The SMA is the least efficient model here, because several of its memory modules are dual ported in order to be accessed by the processing elements.

In Fig. 11, there is a data memory power comparison between the full search and the Hierarchical Search algorithm for the DMA model. Transformations 5 and 13 are optimal for both algorithms, a fact that could possibly allow us to execute both algorithms on the same platform, which would be convenient since they are both parts of the motion estimation kernel in MPEG-4. It can also be seen that in the case of the Full-Search algorithm where the number of the executed operations is large (29.89 GOPS), the application of data reuse transformations has a great impact on the consumed data power in comparison to the original algorithm, partitioned or not. But in the case of the hierarchical motion estimation algorithm, where the number of operations is considerably smaller (507.38 MOPS), the effect of the data reuse transformations is limited. In fact even the role of the particular architecture diminishes and all models seem to converge.

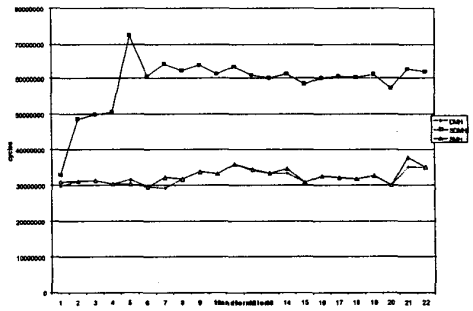

Figure 9. *Comparison results for performance*

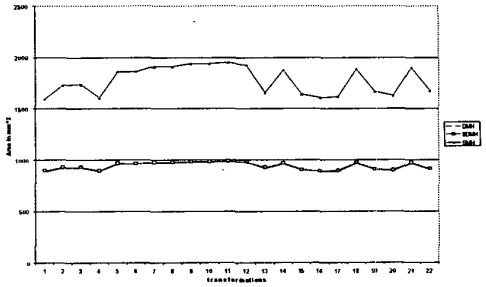

Figure 10. *Comparison results for area*

In Fig. 12 it can be seen that the presence of cache memory results in instruction power savings between 10% and 90% for all architectures. The impact is less great on the SDMA model than on the SMA and DMA. Similar results apply in the case of the Full-Search Motion Estimation Algorithm.

Our experiments clearly state that there is no target architecture that is best for both applications. Clearly the results are algorithm-dependent even for similar algorithms like those presented.

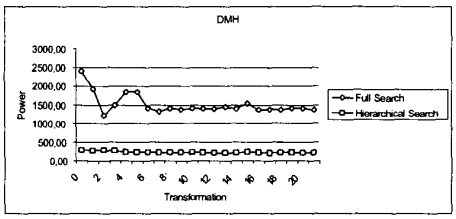

Figure 11. *Comparison between Full-Search and Hierarchical Search*

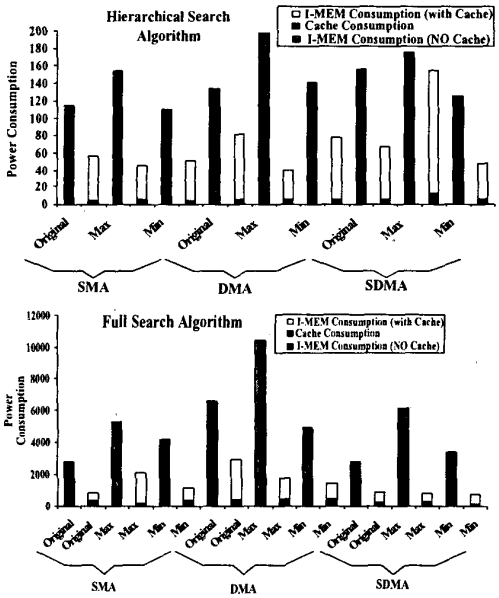

Figure 12. *Cache Analysis for the Hierarchical Search and the Full Search Algorithms*

5. Conclusions- Future work

Memory optimization and partitioning for the widely used Hierarchical Search Motion Estimation algorithm was achieved, with the use of global loop and data-reuse transformations. The results showed that the impact of the data reuse transformations on data power depends on the complexity of the algorithm.

Furthermore, the significant contribution of the instruction memory power in the total power consumption focuses our efforts in finding an efficient way to reduce it. Apart from using performance optimizations, an option would be the insertion of instruction caches (memory hierarchy).

6. References

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

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

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

[4] D.Soudris, N.D. Zervas, et al., "Data-Reuse and Parallel Embedded Architectures for Low-Power, Real-Time Multimedia Applications", in Proc. of PATMOS'00, September 2000, pp.243-254.

[5] V. Bhaskaran and K. Konstantinides, Image and Video Compression Standards, Kluwer Academic Publishers, Boston, 1998.

[6] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Compilers: Principles, Techniques and Tools. Addison-Wesley, 1986.

[7] L. Nachtergaele, B. Vanhoof, F. Cathoor, D. Moolenaar, and H. De Man, 'System-level power optimizations of video codecs on embedded cores: a systematic approach", Journal of VLSI Signal Processing System, Kluwer Academic Publishers, Boston, 1998.

[8] S. Y. Kung, "VLSI Array Processors", Prentice Hall, Eaglewood Cliffs, 1998.

[9] P. Landman, Low power architectural design methodologies, Doctoral Dissertation, U.C. Berkeley, Aug 1994.

[10] ARM software development toolkit, v2.11, Copyright 1996-97, Advanced RISC Machines

[11] 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. SC26, No. 1, Feb 1991, pp.98-105.

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