Data-Reuse and Parallel Embedded Architectures for Low-Power, Real-Time Multimedia Applications

D. Soudris[1], N. D. Zervas[2], A. Argyriou[1], M. Dasygenis[1], K. Tatas[1], C. E. Goutis[2], and A. Thanailakis[1]

[1] VLSI Design and Testing Center, Dept. of Electrical & Computer Eng., Democritus Univ. of Thrace, Xanthi 67100, Greece.
[2] VLSI Design Lab., Dept. of Electrical & Computer Eng., Univ. of Patras, Rio 26500, Greece.

Abstract. Exploitation of data re-use in combination with the use of custom memory hierarchy that exploits the temporal locality of data accesses may introduce significant power savings, especially for data-intensive applications. The effect of the data-reuse decisions on the power dissipation but also on area and performance of multimedia applications realized on multiple embedded cores is explored. The interaction between the data-reuse decisions and the selection of a certain data-memory architecture model is also studied. As demonstrator a widely-used video processing algorithmic kernel, namely the full search motion estimation kernel, is used. Experimental results prove that improvements in both power and performance can be acquired, when the right combination of data memory architecture model and data-reuse transformation is selected.

1 Introduction

The number of multimedia systems used for exchanging information is rapidly increasing nowadays. Portable multimedia applications, such as video phones, multimedia terminals and video cameras, are available. Portability as well as packaging, cooling and reliability issues have made power consumption an important design consideration [1]. For this reason there is great need for power optimization strategies, especially in higher design levels, where the most significant savings can be achieved.

Additionally, these applications also require increased processing power for manipulating large amounts of data in real time. To meet this demand, two general implementation approaches exist. The first is to use custom hardware dedicated processors. This solution leads to smaller area and power consumption. However, it lacks of flexibility since only a specific algorithm can be executed by the system. The second solution is to use a number of embedded instruction set processors. This solution requires increased area and power in comparison to the first solution. However, it offers increased flexibility and mainly meets easier the

D. Soudris, P. Pirsch, and E. Barke (Eds.): PATMOS 2000, LNCS 1918, pp. 243–254, 2000.

time-to-market constraints. In both cases, to meet the real time requirements, the initial application description must be partitioned and assigned to a number of processing elements, which has to be done in a power efficient way. For multimedia applications realized in custom-processor platforms, the dominant factor in power consumption is the one related to data storage and transfer [2]. In programmable platforms though, the power consumed for instructions storage and transfers limits the dominant role of the power related to data storage and transfer [4].

The related work that combines partitioning of the algorithm and techniques for reducing the memory related power cost is relatively small [2][3][4][5]. More specifically, a systematic methodology for the reduction of memory power consumption is presented in [2][3]. According to this methodology, power optimizing transformations (such as data-reuse) are applied in the high level description of the application prior to partitioning step. These transformations mainly targets to reduction of the power due to data storage and transfer. Although, the efficiency of this methodology has been proved for custom hardware architectures [2] and for commercially available multimedia processors (e.g. Trimedia) [3], it does not tackle with the problem when an embedded multiprocessor architectures are used. The latter point has been stressed in [4] where the data-reuse exploration as proposed in [6] has been applied for uni-processor embedded architectures. The experimental results of [4] indicated that the reduction of the data memory-related power does not always come with a reduction of the total power budget for such architectures. Finally, a partitioning approach attempting to improve memory utilization is presented in [5]. However, this approach limited by the two-level memory hierarchy, does not explore the effect of the high-level power optimizing transformations, and its applicability is limited to a class of algorithms expressed in Weak Single Assignment Code (WSAC) form. Clearly, previous research work has not explored the effect on power, area, and performance of the high level transformations for the case of multiprocessor embedded architectures. In such architectures a decision that heavily affects power, area and performance is the one related to the data memory architecture-model (i.e. shared, distributed, share-distributed) to be followed.

The motivation of this work is to investigate the dependencies between the decision of adapting a certain data memory architecture-model and the high-level power optimizing transformations. The intuition is that these two high-level design steps, which heavily influence all design parameters are not orthogonal to each other. Consequently, in this paper we apply all possible data-reuse transformations [6] in a real-life application, assuming a LSGP partitioning scheme [11] and three different data memory architecture-models, namely Distributed, Shared, and Shared-Distributed. For all the data-memory architectures, the transformations' effect on performance, area and power consumption is evaluated. The experimental results prove that the same data-reuse transformations do not have similar effect on power and performance when applied for different data-memory architecture models. Thus, the claim that the application of these transformations in the first step can optimize power and/or performance, regard-

less the decisions related to data memory architecture that must follow is proved to be weak. Furthermore, the comparative study concerning power, performance and area of the three architectures and all the data reuse transformations indicate that an effective solution can be acquired from the right combination of data memory architecture model and data-reuse transformation. Finally, once more, the critical influence of the instruction power consumption on the total power budget is proved.

2 Target Architectures

We are working on multiple processor architectures each of which has its own single on-chip instruction memory. The size of the instruction-memory is strongly-depended on the code size executed by a processor. We name this scheme application specific instruction memory (ASIM). The instruction memory of different processors may have different size. Concerning the data-memory organization, application specific data memory hierarchy (ASDMH) is assumed. [2][7]. Since we focus on parallel processing architectures, we explore ASDMH in combination with three well-established data-memory architectures models: 1)*distributed data-memory architecture DMA*, 2) *shared data-memory architecture SMA*, and 3) *shared-distributed SDMA* data memory architecture. 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. With DMA, a separate data-memory hierarchy exists for each processor (Fig. 1). In this way all memories modules of the memory hierarchy are single ported, but also area overhead is possible in cases of large amount of common data to be processed by the N processors. The second data-memory architecture-model (i.e. SMA) implies a common hierarchy of memory levels for the N processors (Fig. 2). Since, in the data-dominated programmable parallel processing domain, it is very difficult and very performance inefficient to sequentially schedule all memory accesses, we assume that the number of ports for each memory block equals the maximum number of parallel accesses to it. Finally, SDMA is a combination of the above two models, where the common data to the N processors are placed in a shared memory hierarchy, while a separate data memory hierarchy also exist for the lowest levels of the hierarchy (Fig. 3). For experimental purposes, we have considered target models with N=2 without any restriction about memory hierarchy levels.

3 Data Reuse Transformations

The fact that in multimedia applications the power related to memory transfers is the dominant factor in total power cost, motivate us to find an efficient method to reduce them. This goal can be done by efficient manipulation techniques of memory data transfers. For that purpose, we performed an exhaustive data reuse

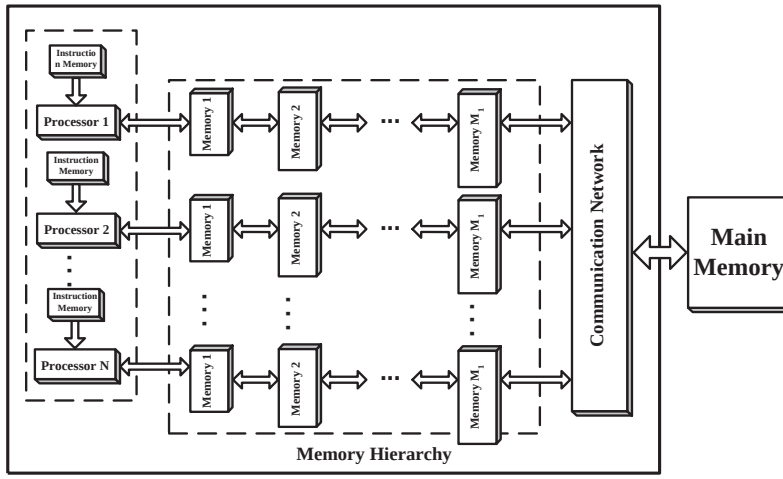

Fig. 1. The distributed memory data-memory architecture model

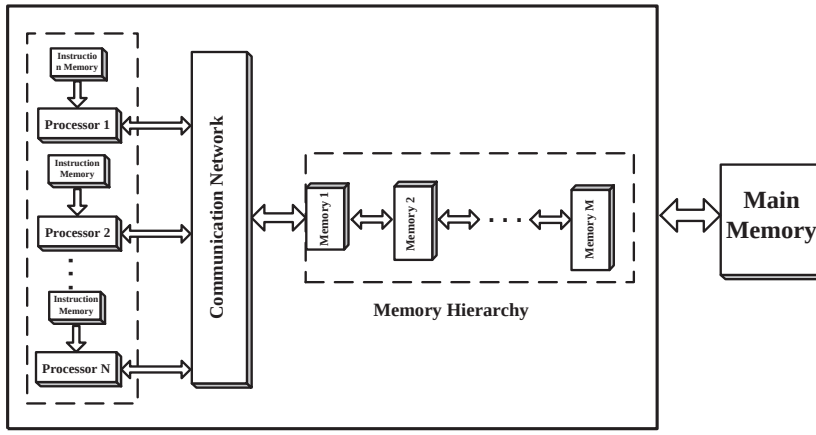

Fig. 2. The shared memory data-memory architecture model

exploration of the application's data. Employing data reuse transformations, we determine the certain data sets, which which are heavily re-used in a short period of time. 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 transfered on chip, reducing power consumption related to data transfers. Of course, data reuse exploration has to decide which data sets are appropriate to be placed in separate memory. Otherwise, we will need a lot of different memories for each data set resulting into a significant area penalty.

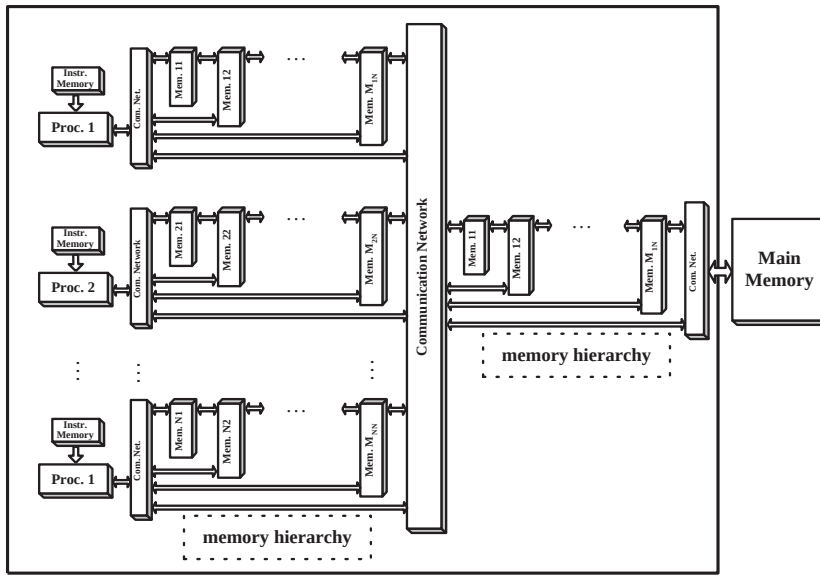

Fig. 3. The shared-distributed data-memory architecture model

Since our target architecture consists of programmable processors, we must take into consideration the power dissipation due to instruction fetching. Previous work [4] forms a sign that this power parameter is a significant part of total system's power, and thus, it should not be ignored. Also, it depends on both number of executed instructions and the size of the application code. Particularly, the number of executed instructions determines how many times the instruction memory is accessed, while the code size determines the memory size. The cost function used for our data reuse exploration on all target architectures is evaluated in terms of power, performance, and area, taking into account both data and instruction memories. The cost function for power is:

$$Power\_cost = \sum_{i=1}^{N} power\_cost_i \tag{1}$$

where N is the number of processors and the i-th power estimate, $power\_cost_i$ is:

$$\begin{aligned} power\_cost_i = \sum_{c \epsilon CT} & [P_r(word\_length(c), \#words(c), f_{read}(c), \#ports(c)) \\ & + P_w(word\_length(c), \#words(c), f_{write}(c), \#ports(c))] \\ & + P_i(instr\_word\_length, code\_size, f) \end{aligned} \tag{2}$$

where c is a member of the copy tree (CT) [6] , $P_r(\cdot)$, $P_w(\cdot)$, and $P_i(\cdot)$ is the power consumption estimate for read operation, write operation, and instruction

fetch, respectively. For memory power consumption estimation we use the models reported in [2]and [8].

The total delay cost function is obtained by:

$$Delay\_cost = \max_{i=1,\ldots,N} \{\#cycles\_processor_i\} \tag{3}$$

where $\#cycles\_processor_i$ denotes the number of the executed cycles of the i-th processor ($i = 1, 2, \cdots, N$). Also, the maximum number of cycles is the performance of the system. 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 [12].

High level estimation implies that a designer should decide, which possible solution of a certain problem is the most appropriate. For that purpose, we will use the measure of *power* $\times$ *delay product*. This measure can be considered as a generalization of the similar concept from circuit level design and allows the designer performing trade-offs among several possible implementations. That is, the power efficient architecture is:

$$Power\_eff\_arch = Power\_cost \times Delay\_cost \tag{4}$$

The corresponding area cost function is:

$$Area\_cost = \sum_{i=1}^{N} area\_cost_i \tag{5}$$

with

$$\begin{aligned} area\_cost_i = & \sum_{c \epsilon CT} Area(word\_length(c), \#words(c), \#ports(c)) \\ & + Area(instr\_word\_length, code\_size) \end{aligned} \tag{6}$$

For the area occupied by the memories, Mulder's model is used [9]. The cost function of the entire system is given by:

$$Cost = a \cdot Power\_eff\_arch + b \cdot Area\_cost \tag{7}$$

where a and b are weighting factors for area/energy trade-offs.

4 Experimental Results-Comparative Study

In this section, we perform extensive comparative study of the relation between data-reuse transformations and data-memory models, assuming the application's partitioning. We begin with the description of our test vehicle and through its partitioning scheme, we will provide the experimental results after the application of the data-reuse transformations for all target architectures, in terms of power performance and area.

4.1 Demonstrator Application and Partitioning

Our demonstrator application was selected to be the full search motion estimation algorithm [10]. It was chosen this algorithm because it is used in a great number of video processing applications. 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. The algorithm structure is described in Figure 4(a) which has three double nested loops. A block of the current frame (outer loop) is compared to a number of candidate blocks (middle loop). In the inner loop, a distortion criterion is computed to perform the comparison.

Partitioning was done with the use of LSGP technique [11]. By applying this technique to a generalized for-loop structure, while assuming p partitions, the form of the partitioned algorithm becomes as shown in Fig.5.

```
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+l) < 0 || (B*y+j+l)>M-1)
            \*conditional statement for the pixel of candidate block * \
```

Fig. 4. The full 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
```

Fig. 5. The partitioned algorithm

The semantic *"Do in parallel"* imposes the parallel (concurrent) execution of p nested loops (i.e. sub-algorithm). From this above -code, it is apparent that the outermost loop is broken into p partitions, each of which is mapped to processor. The p processors execute the same algorithmic structure for different values of loop index x, i.e. different current blocks. Due to the inherent property

of algorithm, a set of data should be used by two consecutive sub-algorithms. In other words, data from (k-1)-th processor should be used by k-th processor ($k = 1, 2, 3, \cdots, p$). Our experiments were carried out assuming $p = 2$, meaning two partitions. Therefore, the loop index x has a range of nine. Due to QCIF format (144x176), the outermost index ranges from 0 to 8. The first and second processor execute the algorithm in parallel fashion, for loop index x ranging from 0 to 4 and from 5 to 8, respectively. We examined the impact of partitioning combined with 21 data reuse transformations on power, performance, and area. These transformations were applied after the partitioning process was finished in accordance with the previous section. 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). These transformations were applied for all the three data-memory architecture modeles by taking into account each architecture's characteristics. In Fig. The copy tree [6]of the full search motion estimation algorithm is identical for processor 1 and 2, where the dashed lines show the memory levels. Each rectangle contains three labels, where the number determines the applied data reuse transformations associated to memory hierarchy level. The remaining two labels determine the size of an PB and CB line or block, RW line or reference window.

4.2 Experimental Results

Comparisons among the three target architectures, in terms of power, performance, and area are shown in Fig. 6,8, and 9.

Fig. 6 provide comparisons results of power consumption with respect to data-reuse transformations. The most power efficient design approach is the combination of SDMA and data-reuse transformations 4,5,15,19 and 20. In contrary, almost all data-reuse transformations increase the total power when DMA or SMA is assumed.

The effect of the data-reuse transformations on power consumption of data memory is shown in Fig. 7. As it can be seen, the largest effect is on the SMA, while the most efficient are the two other two data memory architecture models. Comparing Fig. 6 and 7, it is deducted that the power cost related to instruction memory have significant contribution on the total power budget, and in many cases overturns the power savings acquired in the data memory. Thus, the power component related to instruction-memory cannot be ignored during such high level power exploration. Fig. 8 shows that with DMA and SMA the data-reuse transformations barely affects performance, while with SDMA the transformations have a more significant impact on performance. The greater variation in performance when the SDMA is assumed results from the size of instruction code related to control operations, specifying which memories of the hierarchy should be accessed. However, it can be generally concluded that the transformations have similar effect on the performance for all data-memory architecture models (i.e. a certain transform positively/negativelly affects performance for all data-memory architecture models). Although this is true, the optimal transfor-

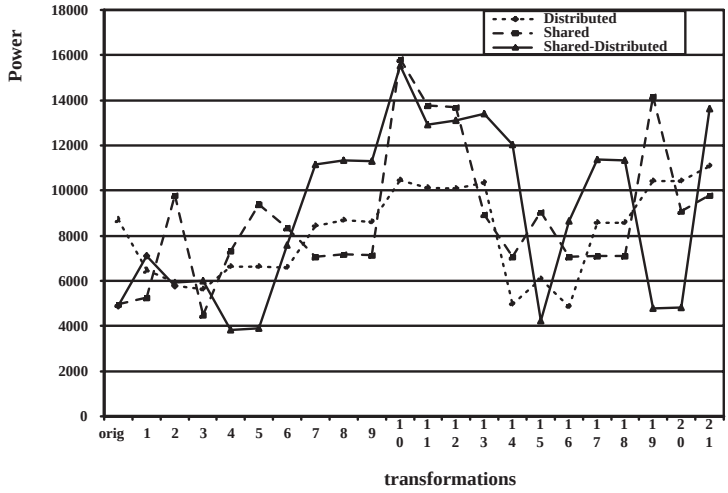

Fig. 6. Comparison results for total power.

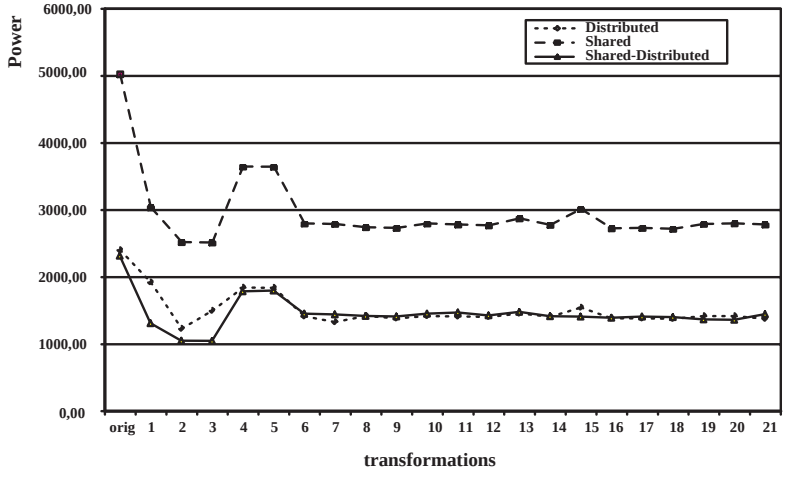

Fig. 7. The effect of data-reuse transf. on power of data memory.

mations in terms of performance are different for each different data-memory architecture model. Specifically 4,5,6,18,19 and 20 for SDMA, 6,7,8,9,13,16,17 and 18 for SMA and DMA are the near-optimal or optimal solutions in terms of performance.

In Fig.9 the effect of data-reuse transformations on area is illustrated. From that it can be inferred that each transformation influences area in almost iden-

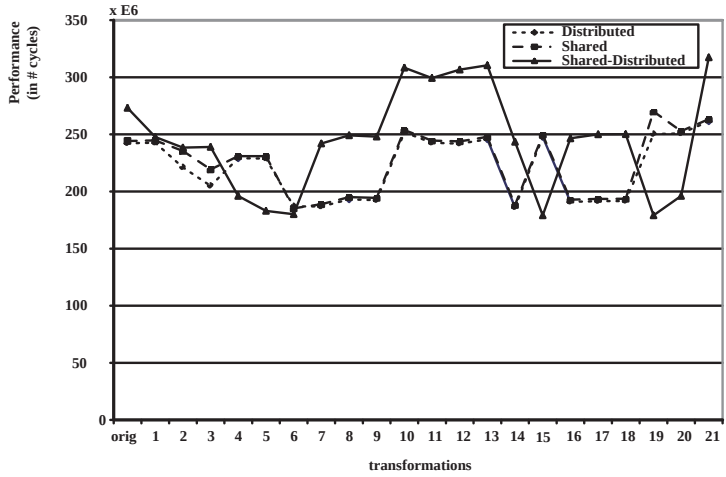

Fig. 8. Performance comparison results of the target architectures.

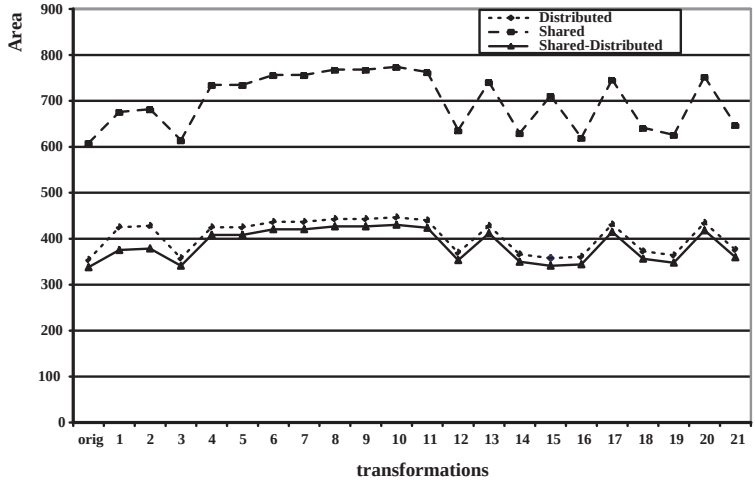

Fig. 9. Area comparison results of the target architectures.

tical manner for all data-memory architectural models. It is also clear that all transformations increase area, since they impose the addition of extra data memory hierarchy levels. Moreover, for both DMA and SDMA area cost is similar for each data-reuse transformation. With SMA the area occupation is larger in all cases. This due to the fact that several memory modules are dual ported, to be accessed in parallel by the processing elements. In contrary, most memory modules are single ported and thus, they occupy less area. As it can be seen,

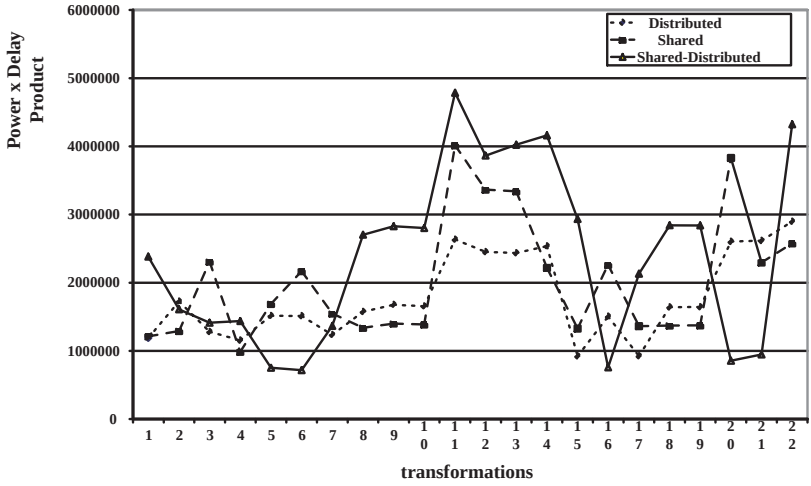

Fig. 10. Comparison results of the target architectures with respect to *power* × *delay product*.

the SDMA is the most area efficient, since with this data-memory architecture model there are no memories in the hierarchy with duplicate data.

In order to define which combination of data-memory architecture model and data-reuse transformation is the most efficient in terms of performance and power, we plot *power* × *delayproduct* (Fig.10). We infer that there exist enough possible solutions, which can be chosen by the designer. These solutions are: the transformation 3 with SMA, transformations 15 and 17 with DMA and transformations 4,5,15,19 and 20 with SDMA. If also the area dimension is taken into account, the effective solutions are transformations 15 and 17, and, 4,5,15,19 and 20 with DMA and SDMA, respectively.

5 Conclusions

Data-reuse exploration for the partitioned version of a real life application and for three alternative data-memory architecture models was performed. Application specific, data-memory hierarchy and instruction memory, as well as embedded programmable processing elements, were assumed. The comparison results prove that an effective solution either in terms of power or power and delay or power and delay and area, can be acquired from the right combination of data memory architecture model and data-reuse transformation. Thus, in the parallel processing domain for multimedia applications, the high-level design decision of adapting a certain data-memory architecture model and the application of high-level power optimizing transformations should be performed interactively and not in a sequential way (regardless the ordering) as prior research work proposed.

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. K. Masselos, F. Catthoor, H. De Man, and C.E. Goutis, and "Strategy for Power Efficient Design of Parallel Systems", in IEEE Trans. on VLSI, vol. 7, No. 2, June 1999, pp. 258-265.
4. 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.
5. U. Eckhardt and R. Merker,"Hierarchical Algorithm Partitioning at System Level for an Improved Utilization of Memory Structures", in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 1, January 1999, pp. 14-23.
6. S. Wuytack, J.-P. Diguet, F. Catthoor, D. Moolenaar, and H. De Man "Formalized Methodology for Data Reuse Exploration for Low-Power Hierarchical Memory Mappings", in IEEE Trans. on VLSI Systems, Vol. 6, No. 4, Dec. 1998, pp. 529-537.
7. L. Nachtergaele, B. Vanhoof, F. Catthoor, D. Moolenaar, and H De Man,"System-level power optimazations of video codecs on embedded cores: a systematic approach", Journal of VLSI Signal Processing Systems, Kluwer Academic Publishers, Boston, 1998.
8. P. Landman, Low power architectural design methodologies, Doctoral Dissertation, U.C. Berkeley, Aug. 1994.
9. 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.
10. V. Bhaskaran and K. Kostantinides, Image and Video Compression Standards, Kluwer Academic Publishers, Boston, 1998.
11. S. Y. Kung,"VLSI Array Processors", Prentice Hall, Eaglewood Cliffs, 1988.
12. ARM software development toolkit, v2.11, Copyright 1996-7, Advanced RISC Machines.