Layer Assignment Techniques for Low Energy in Multi-layered Memory Organisations

E. Brockmeyer, M. Miranda, H. Corporaal*, F. Catthoor†

* Also professor at the Technische Univ. Eindhoven, The Netherlands

† Also professor at the Katholieke Univ. Leuven, Belgium

Abstract

Nearly all platforms use a multi-layer memory hierarchy to bridge the enormous latency gap between the large off-chip memories and local register files. However, most of previous work on HW or SW controlled techniques for layer assignment have been mainly focussed on performance. As a result, the intermediate layers have been assigned too large sizes leading to energy inefficiency. In this paper we present a technique that takes advantage of both the temporal locality and limited lifetime of the arrays of the application for minimum energy consumption under layer size constraints. A prototype tool has been developed and tested using two real-life applications of industrial relevance. Following this approach we have been able to half the energy consumed by the memory hierarchy for each of our drivers.

1 Introduction

Existing platforms nearly always have more than one layer in their memory subsystem. These layers are inserted to bridge the enormous performance, latency and energy consumption gap between the large off-chip memories and the processor. Memory hierarchy layers can contain normal (software controlled) memories or caches. An application has to be mapped efficiently on this memory hierarchy. Often this requires that smaller copies are made from larger data arrays which can be stored in the smaller layers [2]. Those copies must be selected such that they minimize the miss cost of all the layers globally. Any transfer of data from a higher layer to the current one is considered to be a miss for the current layer. This happens most efficiently under software control because a compiler can take a global view. In the case of local memories, copy operations should be explicitly present in the code. However, in the case of hardware controlled caches, the cache controller will make the copies of signals at the moment they are accessed (and the copy is not present in the cache yet). So the code must be written such that the controller is forced to make the right decision [4].

Memory Hierarchy Layer Assignment (MHLA) will take advantage of temporal locality and limited lifetime of the arrays in order to minimize the energy consumption within the constraints. The search space is explored for the minimum energy within timing and memory architecture constraints while taking into account the copy overhead. In current designs, the intermediate layers are not used efficiently and can be made factors smaller consuming less energy while maintaining an equally small miss rate and meeting the performance requirements.

2 Problem statement

By exploiting data reuse, a part of an array is copied from one layer to the lower layer from where it is read multiple times. As a result, energy can be saved since most accesses take place on the smaller copy and not on the large more energy consuming original array. Many different opportunities exist for making a data reuse copy. These are called copy candidates (CCs). Only when it has been decided to instantiate a CC we call it a copy. A relation exists between the size of a CC and the number of transfers from the higher layer, typically called misses (see Fig. 1). This figure shows a loop nest with one reference to an array A with size 250. The array has 10000 accesses. Several CCs for array A are possible. For example we could add a copy A" of size 10 which is made in front of the k-loop by adding the statement "for (z=0; z<10; z++) A2[z]=A[j*10+z];". This copy[1] statement is executed 100 times, resulting in 1000 misses to the A array. This CC point is shown in Fig. 1b. Note that the good spatial locality in this example does not influence the amount of misses to the next level. In theory any CC size ranging from one element to the full array size is a potential candidate. However, in practice only a limited set of CCs leads to efficient solutions. Obviously, a larger CC can retain the data longer and can therefore avoid more misses. All possible CCs are shown in Fig. 1b. The most promising CC sizes and miss counts are kept and put into a data reuse chain as shown in Fig. 1c. These are exactly those that have a relation to the loop bounds. This data reuse chain is completed with the 250 writes to the array. The above example considers only a single array with one reference. In practice multiple arrays exist, each with one

[1] Though the copy candidates will be instantiated as arrays in the application we reserve the name array for the "original array".

1530-1591/03 $17.00 © 2003 IEEE

or more references. To each read reference corresponds a reuse chain. These chains are combined in a reuse tree. For example, the upper left of Fig. 2 shows two data reuse trees (array A has 2 references). Indeed, the second reference of A has no promising CC. More details on identification of data reuse chains and trees can be found in [2, 12].

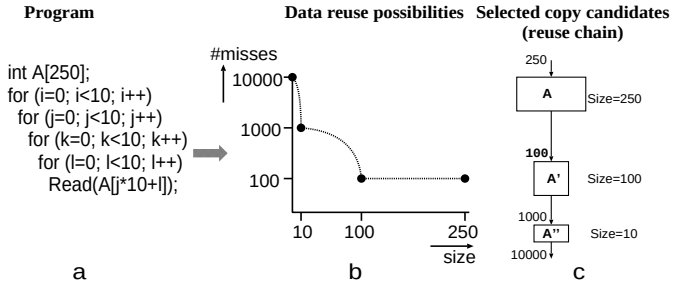

Figure 1. Data reuse information

In the next step, CCs and arrays are mapped to a data memory hierarchy. We consider a generic target platform. It contains multiple memory layers L_i, each layer contains multiple memory partitions, where each partition contains multiple memory modules. All memory modules within a partition are of the same type but can have different sizes and number of ports. Typical types are software controlled SRAM or DRAM (typically called scratchpad memories), off-chip (S)DRAM and caches.

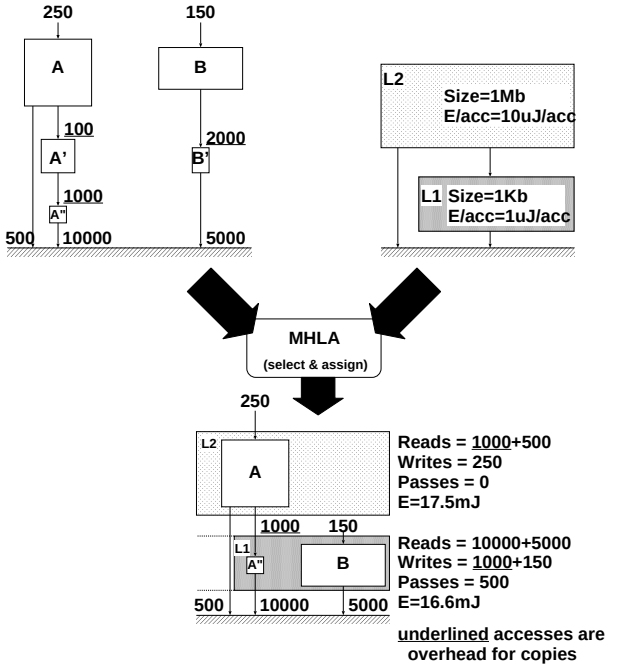

Figure 2. MHLA problem definition

Now we are capable of defining the **MHLA problem**: *MHLA=minimizing the energy consumption of the data memory hierarchy for a given program and platform by selecting a set of CCs and assigning them together with the arrays to the memory partitions within the layers of the platform.*

MHLA determines the energy consumption of a mapping by calculating the activity of the individual partitions and using the energy consumption per access. This cost is a function of size and other memory parameters and is modeled in a memory library. The MHLA process and its mapping result are depicted in Fig. 2. MHLA has selected the A" and B to be stored in L1 and the A array in L2. As a result 250 writes occur on L2 for A, 500 reads for the first access and 1000 misses for A". The L1 layer has 150 writes for the B array, 1000 writes due to the misses of A" and 15000 reads for both A" and B. Note that the 500 accesses of the first A reference do not affect the activity of the L1 for this architecture. This may not be the case for hardware controlled caches. Because all accesses have to pass through the cache when no bypass is foreseen. Also note that for caches no explicit copies are introduced in the code. However, the cache controller can be enforced to make the desired copy by a proper memory layout [4].

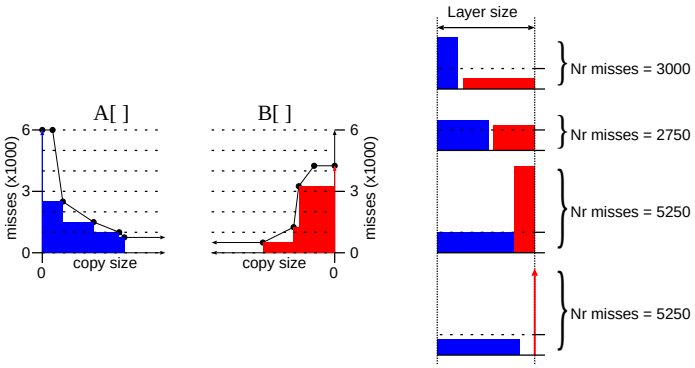

Figure 3. Trading of the copy candidate size

The mapping of arrays and CCs must be performed globally in order minimize the energy consumption. The size of one copy must be traded for the size of another copy because layer size is constraint or must be kept low for energy reduction. A simple illustrative example of the data reuse tradeoff having multiple references is given in Fig. 3. The two left hand side curves show the miss count versus copy size tradeoff for two access references in an application. The access to array A has a maximum of 6000 misses which can be reduced below 1000 misses when the largest interesting CC is selected. Similarly, the array B has about 4000 misses without CC and 500 for the largest CC. The number of misses is minimal when selecting the largest CC for both. However, this assignment leads to an infeasible implementation as the total size does not obey the layer size constraint given at the right hand side of Fig. 3. The column at the right hand side of this figure shows the feasible combination of CCs in this layer. The upper solution combines the largest CC of B and the largest feasible CC of A. The to-

tal number of 3000 cache misses is the sum of the individual CCs. While constructing the other solutions, some space of the B copy candidate is traded for the A copy candidate. As a result the number of misses for A reduces and the number of misses of B increases. In this simple example it is easy to determine the optimal solution having 2750 misses. In general however, realistic applications have too many possibilities to still find the best solutions manually. Certainly when trading the selection and assignment of CCs together with the assignment of arrays to a layer. This is needed in order to avoid unnecessary copying from one layer to the next. Additionally the problem must be considered over the memory hierarchy globally because misses of one layer can be traded for misses on another layer. So an automated technique and tool is needed.

3 Related work

Optimizing the memory hierarchy for performance is a well explored topic [3, 7, 11, 6]. Also several recent papers address the energy related issues [9, 12].

The characterization of [14] clusters the data sets into different types. These different clusters have certain memory type preferences and are assigned accordingly. The mapping is sub optimal (especially for the regular accesses) because it is based on some average characteristics and does not allow and accurate prediction. Afterward, the performance is measured by simulation. Also [15] made a distinguish between caches and scratchpad memories. However, no real layer assignment was made. The technique presented in [18] assign data (and instructions) to the scratchpad. However, no consideration is made to benefit from reuse and inplace opportunities.

The closest work is presented in [12]. It analyses and exploits the temporal locality by inserting local copies. Their layer assignment builds a separate hierarchy per loop nests and then combines them into a single hierarchy. However, a global view of the assignment and lifetime of the arrays and copies is required for real life applications having imperfect nested loops. Moreover, no overhead estimation is made which makes it impossible to tradeoff copies versus arrays in a certain layer. Similarly, the work published [8] lacks of a global application view.

Access trace based analysis techniques like presented [10] have limited optimization capabilities. The quality of the analysis depends on the preceding compilation step. For instance, from access profile point of view all elements of an array are accessed equally while a small data reuse copy could be present. As a result, the exploration space cannot be searched properly.

To our knowledge no one has combined the opportunities for data reuse and inplace in an automatable technique for real life application that is not based on simulation. Our approach allows finding the optimal assignment in a predictable way for both memories and hardware caches.

4 Exploration methodology

An efficient mapping of arrays and CCs to partitions must be found within reasonable time. The exploration is performed in two phases. The first phase assigns all the arrays to the partitions. Indeed, all arrays have to be assigned to guarantee functionality. In the second phase, a selection of CCs is mapped to the remaining partition space for each valid array assignment. In principle we can fully search all possible mappings of arrays and CCs. This may take several hours though. Therefore we have implemented a steering heuristic and have optimized the implementation by so called incremental assignment.

Steering heuristic: The arrays having the highest access over size ratio are assigned to the cheapest possible partition first. Intuitively this makes sense, as then the cheapest partitions will get most accesses. A similar heuristic is used for the CCs. However, for CCs we do not know the number of accesses before all possible CCs are mapped. For example, in Fig.1 the number of reads to A' is 1000 or 10000 depending on whether A" is mapped or not. Therefore we use the highest ratio of misses over CC size. This is also logical because these copies avoid a larger amount of misses.

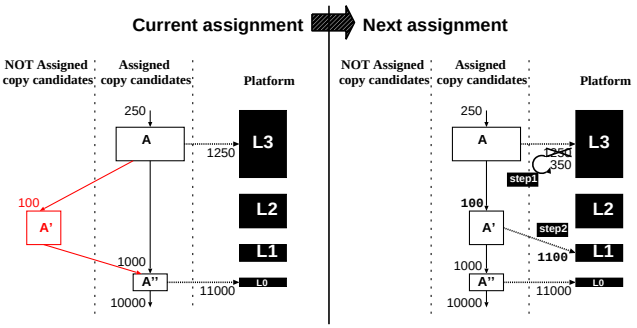

Figure 4. The next assignment is incrementally calculated

Incremental assignment: The MHLA exploration works in an incremental way for exploration efficiency. From a given, current assignment a new one is constructed; see Fig. 4. The left hand side shows an example of the current assignment where the array A is assigned to L3, A" to L0 and the copy candidate A' unassigned. First a new CC is selected. In the example, A' is the only CC remaining and is selected. This CC is assigned iteratively to those memory partitions that are in between the layers to which the higher and lower CC are assigned. In Fig. 4, A' can only be assigned to partitions in L1 and L2. Other partitions do not have to be searched for. Independent of the partition assignment of A', the misses 1000 of A" do not take place anymore on L3 but are replaced by the 100 misses of A'.

If A' is assigned to L1, the 100 misses of A' and the 1000 misses of the next lower assigned CC (A") are added to L1. Furthermore the size impact of the assignment to L1 must be evaluated. The most simple size estimation adds the CC size to its assigned partition. This will lead however to a very poor layer usage. An improved estimation based on limited lifetime is explained in Section 6. The *changes* in size, accesses and the effects on the total energy consumption can be calculated without knowing all details of the already assigned CCs.

5 Design flow

MHLA is part of a design flow that optimizes the data memory hierarchy. Two phases can be distinguished in this flow. First a platform independent phase performs program transformations in order to improve locality, data reuse opportunities and required storage size. If this phase is skipped the approach still works but the final results will typically be worse due to the worse temporal locality. The second phase maps the application to the target platform. This is performed in four decoupled steps. First MHLA determines for each data set a layer and type. Afterwards we have more detailed timing information about the array references. This is required for the following step that optimizes the required memory access parallelism for meeting the timing constrains. Techniques like [19, 13] could be used here. Certain accesses must happen in parallel in order to meet the target budget. The conflicting memory accesses must be either stored in different memories or in a dual port memory. The third step, memory allocation and assignment, assigns the data within the memory partition to the various memories obeying the required parallelism [10, 16]. A final data layout step decides on the storage layout of the data inside the memory. This allows for instance to minimize the required storage size (see next section). At the MHLA step, it is important to estimate the later data layout step. Especially the storage size of the next section has a large impact.

6 Partition size estimation

The exploitation of limited lifetime allows to have smaller layers or to store more data in an equal sized layer. Both can have a huge impact on the (energy) performance. Especially, the short lifetime of the CCs should be considered carefully. Also, it can be expected that a technique without inplace estimation is useless for larger applications. All arrays will have relatively a smaller lifetime when the application size increases.

Fig. 5 explains how we can exploit limited life times and reuse locations. The left part shows how elements of A and B are used in time. The shaded areas indicate which elements are used and for how long. Clearly, the declared size of array A can be reduced by a factor three by reusing the same locations. This is called intra inplace [5]. Furthermore the lifetime of the elements of array B do not overlap with A. Therefore B can also reuse the locations of A. This is called inter inplace [5]. The result of both inplace opportunities is shown on the right of Fig. 5.

We have implemented a low complexity inter inplace estimation. The storage size estimation is based on the simultaneous alive data in the most inner loops of the application. As a result we only have to update the storage size of those inner loops that span the lifetime of the CC.

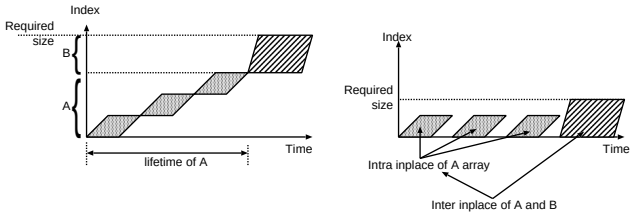

Figure 5. Inplace concept explanation

7 Case studies

Two demonstrators having different characteristics are selected to present the impact of MHLA. The first demonstrator from the video compression fields works mainly on two dimensional arrays and has a lot of data reuse. The second demonstrator is a wireless receiver with limited reuse and only single dimensional arrays. It is worth noting that both algorithms span several pages of complex C code.

Initially we assume the architecture to have two layers. The used partition energy model is based on a real memory model and is displayed by the solid line in Fig. 7. Relative energy figures are sufficient for the tool explore on. All the memory model numbers are relative to the fixed size off-chip memory of 1Mbyte. The largest on-chip memory of 16Kb is a factor 3 less energy consuming than off-chip memory. This conservative factor is realistic and surely not in favor of the method. The energy model is slightly super logarithmic so a memory which is 256$\times$ larger consumes 8.6$\times$ more energy per access. This same energy model is used for L0 and L1 in both drivers. The energy consumption is computed by multiplying by the memory activity. The memory activity is obtained by executing an instrumentation version of the source code.

7.1 QSDPCM

The Quadtree Structured Difference Pulse Code Modulation (QSDPCM) algorithm is an inter-frame compression technique for video images. It involves a hierarchical motion estimation step, and a quadtree based encoding of the motion compensated frame-to-frame difference signal [17].

A global view of the QSDPCM main signals and their reuse is given in Fig. 6. Many data reuse opportunities ex-

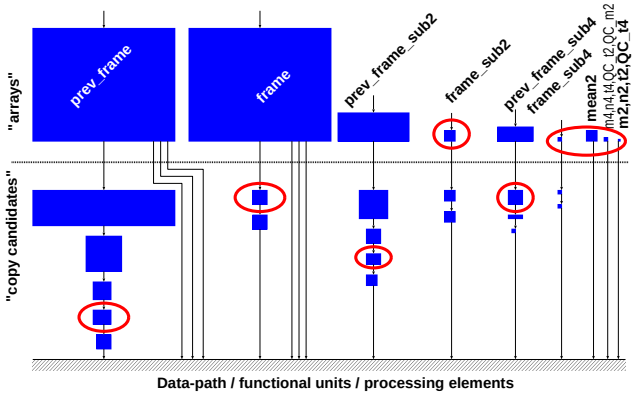

Figure 6. Assignment to memory hierachy.

ists for the QSDPCM application as can be seen from the many data reuse chains. Different runs of the MHLA tool explore the L1 size. Fig. 7 shows the energy contribution of the L1 (bottom bar) and main memory (top bar). When increasing the layer size, the energy goes down because fewer accesses occur on the more energy consuming main memory. The L1 miss rate reduction does not decrease much further for a L1 size larger than 640. Therefore the L1 energy per access increase penalty is not compensated by the lower amount of misses. Hence the overall energy consumption increases. The optimal assignment corresponding to this optimum is given in Fig. 6. The circled arrays and CCs are stored in the L1. The other arrays are stored in the main memory. The not circled CCs are not selected. Inserting an extra smaller third layer did not significantly reduce the energy, as the L1 in the two-layer optimum architecture is already small. When we compare the presented technique to an array assignment technique (crosses) we gain a factor 2 in energy. When switching of the inplace estimation (triangles) we require a L1 of 1K instead of 640 elements to reach the minimum energy. Moreover, it consumes 3.2% more energy.

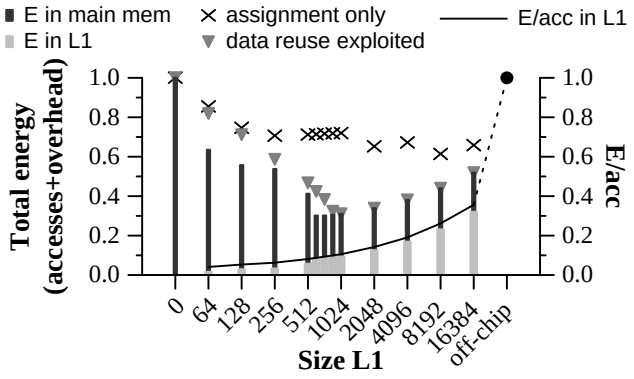

Figure 7. Energy for varying L1 size.

7.2 DAB

Digital Radio is a typical application for portable usage. Therefore low energy issues are very important to extend the battery lifetime. The DAB channel coding standard involves several complex data-dominant algorithms.

Similar to the previous driver, a L1 size exploration is performed and shown in the top curve of Fig. 8. The minimum energy is consumed for a L1 size of 8K elements. The relatively large decrease in energy while increasing the L1 size from 0 to 64 reveals that it might be interesting to insert an L0 of size 64. Therefore we repeat the experiments with an additional layer of size 64 and 128. The additional energy reduction is 15% for both layer sizes at the optimal point. Deeper investigation showed that the additional gain was obtained in the FFT function. Up to 25% of the L1 accesses were removed due to this additional layer.

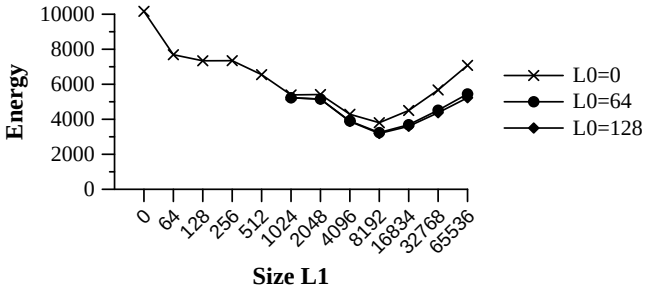

Figure 8. Architecture exploration for DAB.

The most energy efficient assignment has been used to map the DAB application to the TriMedia processor. This processor is selected because it has a data memory hierarchy that matches the optimal architecture. The processor has an L0 layer of 128 registers, 16Kb cache and 8Mb of SDRAM. Importantly, the exploitation of the L0 register file has to be careful evaluated. On one hand, the number of data load stores will reduce because more data remains in the L1. On the other hand, the higher register pressure might counteract this gain as register spilling is required to schedule all instructions. Also the required unrolling, in order to keep the data in the register file, needs more instruction cache space. The tradeoff between the load stores, spilling and instruction cache is given in Table 1. The native TriMedia simulator is used for the evaluation of three differently transformed implementations having a more or less aggressive L0 register usage (second column). The large reduction of 34% in memory accesses has of course a large impact on data memory consumption and performance. Also very important to note, the prediction of the MHLA is very close to the actual number of accesses made by the compiled code. The small difference between the MHLA predicted activity and simulation results are largely explained by the few register spillings and some low level compiler details.

	#reg	#ld/st	pred #ld/st	data cache misses	cycles
Standard reg. usage	6	—	23532	—	—
Mild reg. usage	22	18408	17152	2895	91552
Aggressive reg. usage	70	11837	11232	763	47341

Table 1. DAB results on TriMedia simulator

7.3 Execution time measurements

The tool exploration time is an important factor next to the quality of the results. Table 2 shows the number of explored assignments before finding the optimal solution in comparison to the total number in the huge exploration space. The last column clearly shows that the chosen heuristic allows to find the best solutions within reasonable time. Interesting to note is that the current implementation makes about 20000 evaluations per second on a Pentium-IV.

Size L1	nr. explored assignments	nr. valid assignments	Optimum in # iterations
64	46951	4767	3333
128	351819	45976	34027
256	2563636	540651	1514
512	934606	6175295	9279
1024	20711631	12356321	1077
2048	25552456	23460160	786
4096	28311552	28311552	12

Table 2. Tool performance on QSDPCM

8 Conclusion and future work

This paper has presented the first automated technique to perform layer assignment taking reuse and inplace into account. A fast exploration technique and heuristic is proposed. This technique is implemented in a prototype tool that has allowed us to do exploration on real life demonstrators of industrial relevance. The energy is more than halved by exploiting limited lifetime and reuse in arrays.

The intention is to extend the technique such that it can handle more memory types. The currently supported types are both software and hardware controlled memories. Currently we are adding cycle budget estimation such that it trades energy for performance. Limited data dependent conditions and data dependent addressing is supported. Further research is required to remove more dynamic and data dependent limitations.

References

[1] Radio broadcasting systems; digital audio broadcasting to mobile, portable and fixed receivers. Standard RE/JTC-00DAB-4, ETSI, ETS 300 401, May 1997.

[2] T. Achteren, R.Lauwereins, and F.Catthoor. Systematic data reuse exploration techniques for non-homogeneous access patterns. In *Proc. 5th ACM/IEEE Design and Test in Europe Conf. (DATE)*, pages 428–435, Paris, France, Apr. 2002.

[3] e. a. C.Ancourt. Automatic data mapping of signal processing applications. In *Proc. Intnl. Conf. on Applic.-Spec. Array Processors*, pages 350–362, Zurich, Switzerland, 1997.

[4] C.Kulkarni. *Cache optimization for multimedia applications*. Doctoral dissertation, ESAT/KUL, Belgium, 2001.

[5] E. Greef. *Storage size reduction for multimedia applications*. Doctoral dissertation, ESAT/KUL, Belgium, Jan. 1998.

[6] H-B.Lim and P-C.Yew. Efficient integration of compiler-directed cache coherence and data prefetching. In *Proc. Intnl. Parallel and Distr. Proc. Symp.(IPDPS)*, pages 331–339, Cancun, Mexico, May 2000.

[7] J.Anderson, S.Amarasinghe, and M.Lam. Data and computation transformations for multiprocessors. In *5th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 39–50, Aug. 1995.

[8] e. a. K.Masselos. Memory hierarchy layer assignment for data re-use exploitation in multimedia algorithms realized on predefined processor architectures. In *The 8th IEEE Intnl. Conf. on Electronics, Circuits and Systems (ICECS)*, pages 285–288, Oct. 2001.

[9] L.Benini, A.Bogliolo, and G. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Trans. on VLSI Systems*, pages 299–, 2000.

[10] L.Benini, L.Macchiarulo, A.Macii, and M.Poncino. Layout-driven memory synthesis for embedded system-on-chip. *IEEE Trans. on VLSI*, pages 96–105, 2002.

[11] M.Kampe and F.Dahlgren. Exploration of spatial locality on emerging applications and the consequences for cache performance. In *Proc. Intnl. Parallel and Distr. Proc. Symp.(IPDPS)*, pages 163–170, Cancun, Mexico, May 2000.

[12] M.Kandemir and A.Choudhary. Compiler-directed scratch pad memory hierarchy design and management. In *39th ACM/IEEE Design Automation Conf.*, pages 690–695, Las Vegas NV, June 2002.

[13] P.Grun, N.Dutt, and A.Nicolau. Mist: an algorithm for memory miss traffic management. In *Proc. IEEE Intnl. Conf. on CAD*, pages 431–437, Santa Clara CA, Nov. 2000.

[14] P.Grun, N.Dutt, and A.Nicolau. Apex: access pattern based memory architecture exploration. In *The 14th International Symposium on system synthesis*, pages 25–32, Montreal, Canada, Oct. 2001.

[15] P.R.Panda, N.D.Dutt, and A.Nicolau. Data cache sizing for embedded processor applications. In *Proc. 1st ACM/IEEE Design and Test in Europe Conf. (DATE)*, pages 925–926, Paris, France, Feb. 1998.

[16] P.Slock, S.Wuytack, F.Catthoor, and G. Jong. Fast and extensive system-level memory exploration for atm applications. In *Proc. 10th ACM/IEEE Intnl. Symp. on System-Level Synthesis (ISSS)*, pages 74–81, Antwerp, Belgium, Sept. 1997.

[17] P.Strobach. Qsdpcm – a new technique in scene adaptive coding. In *Proc. 4th Eur. Signal Processing Conf. (EUSIPCO)*, pages 1141–1144, Grenoble, France, Sept. 1988.

[18] S.Steinke, L.Wehmeyer, B-S.Lee, and P.Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *Proc. 5th ACM/IEEE Design and Test in Europe Conf. (DATE)*, pages 409–415, Paris, France, Apr. 2002.

[19] S.Wuytack, F.Catthoor, G. Jong, B.Lin, and H. Man. Flow graph balancing for minimizing the required memory bandwidth. In *Proc. 9th ACM/IEEE Intnl. Symp. on System-Level Synthesis (ISSS)*, pages 127–132, La Jolla CA, Nov. 1996.