Low Power Memory Storage and Transfer Organization for the MPEG-4 Full Pel Motion Estimation on a Multimedia Processor

Erik Brockmeyer, Lode Nachtergaele, Francky V. M. Catthoor, *Member, IEEE,* Jan Bormans, *Member, IEEE,* and Hugo J. De Man, *Fellow, IEEE*

***Abstract*— Data transfers and storage are crucial cost factors in multimedia systems. Systematic methodologies are needed to obtain dramatic reductions in terms of power, area and cycle count. Upcoming multimedia processing applications will require high memory bandwidth. In this paper, we estimate that a software reference implementation of an MPEG-4 video encoder typically requires five Gtransfers/s to main memory for a simple profile level L2. This shows a clear need for optimization and the use of intermediate memory stages. By applying our ACROPOLIS methodology, developed mainly to relieve this data access bottleneck, we have arrived at an implementation which needs a factor 65 less background accesses. In addition, we also show that we can heavily improve on the memory transfers, without sacrificing speed (even gaining about 10% on cache misses and cycles for a DEC Alpha), by aggressive source code transformations.**

I. Introduction

NEXT generation multimedia systems impose heavy demands on the data transfer and storage subsystem [27], [35]. To communicate and hold the massive amounts of data that represent media, fast busses and large memories with high access rates from/to processors are needed. Efficient implementation of the complex media algorithms requires a global analysis of the critical sections and code transformations to eliminate or at least alleviate the impact of these bottlenecks.

The recent MPEG-4 standard [33] is a key to multimedia applications. It involves complex data-dominant algorithms. A hardware or even an embedded software realization of such a (de)coder has to be power efficient in order to reduce the size of the chip packages (where it is embedded) or the battery (if used in a mobile application). It is well-known by now that any future complex chip realization has to take power reduction into account [35]. Our previous research shows clearly the dominant power contribution of data transfer and storage of multidimensional (M-D) array signals and other complex data types in data-dominated designs [6], [27] such as MPEG-4.

In this paper we have exploited this feature to achieve large savings in the system power of a crucial part of MPEG-4, without sacrificing on the performance or on the system latency. The results support our claim that data transfer and storage exploration (DTSE) and optimization for multimedia algorithms have to be performed aggressively before the algorithms are realized in hardware and/or embedded software.

The MPEG-4, multimedia (TriMedia) processor context and the related work are explained in the following Sections II, III and IV. In the next two section, the used memory power model and the MPEG-4 profiling data will be discussed. Section VII and VIII will explain two major steps of our methodology being global loop transformations and the data reuse step. In which the motion estimation kernel will be used as an example. The main topic (in Section IX), will be the transformation of the motion estimation code within a group of VOP's to reduce the memory power. This section also includes an estimation for the number of accesses and a comparison to measured results. All the work so far assumes a software controlled cache. In Section XI, the gain of our methodology is analyzed for a hardware controlled cache, in the case of an H.263 decoder.

II. MPEG-4 Motion Estimation Context

The purpose of the MPEG-4 Video Verification Model (VM) is to describe completely defined encoding and decoding "common core" algorithms and to allow the conduction of experiments under controlled conditions in a common environment [33]. The exploration in this paper has been performed on the MoMuSys Video VM Version 7.0 [30].

The MPEG-4 standard enables an efficient coded representation of the audio and video data that can be "content based," with the aim to use and present the data in a highly flexible way. Every object is coded on its own: the decoder can scale, place and extract the objects from different sources. The size, position and content of the object are variable during a sequence. The video object planes (VOP's), containing coded video sequences and shape information, are divided in MB's (MacroBlock: a group of 16×16 pixels). To exploit the temporal redundancy of a sequence, a H.263 like motion estimation is used. The arrows in Fig. 1 represent all motion estimation steps for one group of VOP's.

The original source uses the baseline, well known, full search motion estimation to generate the motion vectors (MV). The motion vectors can code the information more efficient by using the temporal redundancy and constructing the next VOP out of the previous VOP. All VOP's are divided in MacroBlocks (MB $= 16 \times 16$ pixels), the MB's are sequentially executed in the motion estimation.

Manuscript received September 9, 1998; revised January 6, 1999. The associate editor coordinating the review of this paper and approving it for publication was Prof. Jan-Ming Ho.

The authors are with the Katholieke Universiteit Leuven, Leuven, 3001 Belgium (e-mail: brockmey@imec.be).

Publisher Item Identifier S 1520-9210(99)04097-3.

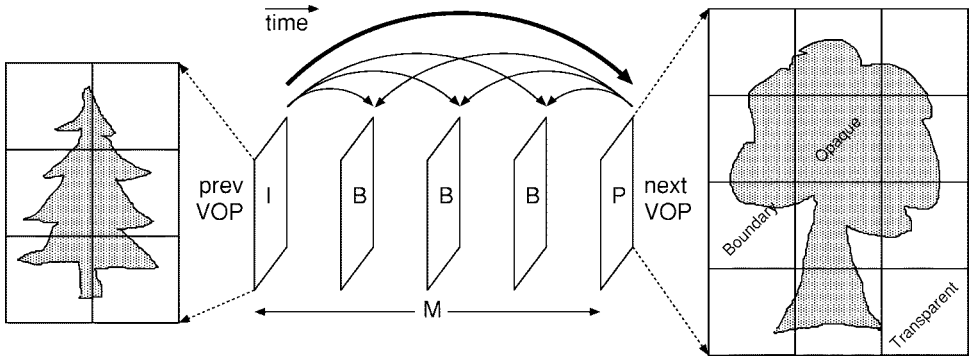


Fig. 1. MPEG-4 sequence, content-based object coding.

A full-pel full search motion estimation calculates, for all MB's of the next VOP and at every possible position of the motion vector in the previous VOP, a sum of absolute differences (SAD) to determine the best match. Depending on an external parameter, the MV length can be limited to 16, 32, 64, $\cdots$ 2048. However, the width and the height of the search area is twice the MV limitation since the MV can point in all directions.

To support objects, an alpha plane is added to every VOP. The alpha plane is a bitmap which indicates which pixels are inside the shape. Since the VOP is divided in MB's, three types of MB's are possible: transparent MB (totally outside the shape), opaque MB (totally inside the shape) and boundary MB (some pixels inside the shape). Only the pixels inside the shape are used during SAD calculation and motion estimation. To avoid a shape mismatch, between previous and next VOP, the previous VOP luminance pixels at the shape edge are repeated outwards (padding and extended padding).

III. Related Work

Up to now, the architecture realizations for state of the art video en/decoders have been focussed on MPEG1-2 and H.263 [21], [26], [27]. We are not aware of published results on full MPEG-4 video encoding yet. The MPEG-4 motion estimation algorithm is based on the H.263 functionality but the embedding in a different context (VOP streams) and other small differences make the current H.263 realizations not directly reusable for efficient realizations. This will be illustrated by the results below, which differ substantially from our previous work on H.263 [27] with the same overall methodology.

Many software-oriented memory management approaches exist in literature but they do not focus on the combination of performance and overall power, which is vital as motivated above. Several papers have analyzed memory organization issues in processors, like the processor and memory utilization [17], [36]. This is however only seldomly resulting in a formalizable method to guide the memory organization issues. The few existing methodologies are usually addressing the "foreground" memory organization issues, i.e., how scalar data is organized in the local register files. An example of this is a theoretical strategy to optimally assign scalars to register-file slots [3]. Some approaches address the data organization in processors for programs with loop nests. Examples include a quantitative approach based on life-time window calculations to determine register allocation and cache usage [5], and work on vector register allocation [1].

In the parallelizing compiler community work has also focussed on loop transformations to improve the locality in individual (regular) loop nests [18], e.g., at Cornell [23], at Illinois [32], and at Stanford [2]. These do not work globally across the entire system, which is required to obtain the largest impact for multimedia algorithms. Partitioning or blocking strategies for loops to optimize the use of caches have been studied in several flavors and contexts, in particular at Hewlett-Packard (HP) [16] and at the University of Toronto [22], [25]. More recently, also multilevel caches have been investigated (see, e.g., [20]). However, the main focus of these memory related transformations has been on performance improvement in individual loop nests though and not on overall power savings or on global algorithms.

In terms of hardware-oriented memory management, most approaches focus on scalar signals [34]. The main differences with our approach are that we can handle large M-D signals within irregular control and loop constructs as required by the MPEG-4 context. The main exceptions are the Phideo approach [24] which is oriented to area optimization in a periodic stream context, and our earlier Atomium work which are focused on full custom architectures [6], [9].

IV. Philips TriMedia Multimedia Processor

The used methodology adapts the source code of the application to a predefined memory architecture. By applying advanced global loop and data reuse transformations the data traffic between the different memory layers can be optimized [10]. To this end accurate knowledge is needed of the target memory architecture.

The application discussed here is the MPEG-4 encoder and the targeted multimedia processor we have used here is the Philips TriMedia. Many of the principles discussed below should remain valid also for other multimedia processors however. Multimedia applications need huge amounts of data and much arithmetic processing so multimedia processors require specialized hardware to meet the high bandwidth and throughput requirements. The TriMedia has several specialized hardware units for multimedia applications, like a receive and send unit for frame oriented data, in parallel and independent from the VLIW-processor core.

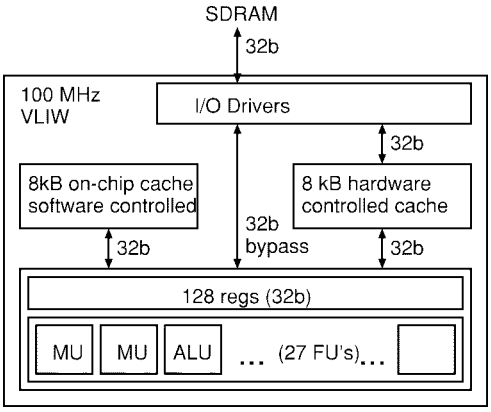


Fig. 2. TriMedia memory architecture.

Multimedia applications are (mostly) data dominant, so the high bandwidth requirement can be reached only if the data stays as much as possible on-chip. For the TriMedia, 128 registers, 16 kB on-chip data cache and 32 kB on-chip instruction cache are available. The 32 kB available instruction cache will not be taken into account because the power contribution of this block can be heavily reduced by hardware solutions for loop dominated multimedia applications [12]. Fig. 2 shows the used memory architecture of the TriMedia [37].

At least half the data cache is hardware controlled, the other half (maximum 8 kB), can be locked to a certain region of the address space and can be "software controlled." The locked cache is modeled as on-chip memory.

To use the locked cache effectively, the programmer (or compiler) has to control the on-chip memory in order to fetch the necessary data only. At compile-time all the knowledge of the algorithm can be used to optimize the (on-chip) RAM usage. Since we mainly optimize for big signals, the smaller signals and the scalars and fully data dependent large signals which can only be resolved at run-time, are taken care of by the hardware controlled cache.

V. Memory Power Model

For data intensive applications, such as video encoding, data transfers dominate the power consumption. Therefore the primary design goal is to reduce memory transfers between large frame memories and data paths. The cost of a data transfer is a function of the memory size, memory type, and the access frequency F_{access}. F_{access} is defined as the real number of accesses per second and *not* the clock frequency. When there is a clock tick and the memory is not accessed, it is assumed that the memory is in power-down mode. This assumption holds for most modern low-power RAM's [19]. The memory itself is characterized by the number of ports, words, bits, and the aspect ratio of the layout. An accurate model, derived from a VTI data sheet, is used for the power exploration in this paper. See [10] for more details.

VI. MPEG-4 Profiling

Here, we will focus only on the most critical part of the MPEG-4 encoder. To get a clear view where the major signal accesses take place, we have instrumented the source code and have run the first 28 VOP's of the "hal" sequence at CIF resolution (group size $M = 8$). Table I shows the number of accesses to the VOP frames for different part of the algorithm. The reference stream VOP average size is small, it contains in total 452 MB's inside the bounding box of which 163 MB's are transparent (in total 39 486 pixels inside the shape). A huge memory bandwidth of 260 M transfers per second is needed for real-time encoding of this MPEG-4 stream at a rate of 30 frames per second. Moreover, the needed bandwidth will increase dramaticly when the VOP becomes bigger. Currently, over 800 k main memory accesses are needed per nontransparent MB. When using a simple profile level L2 the encoder should be able to encode 5940 MB per second [31], which leads to a main memory bandwidth of 5 G. This clearly shows a need for optimization and the introduction of intermediate memory stages.

TABLE I
Number of Reads for the Three Partitions

	#R	#W	#access	%access
YUV motion estimation/ compensation	171M	14M	185M	77%
shape motion estimation/ compensation and coding	34M	16M	50M	21%
YUV coding	3M	2M	5M	2%
total	208M	32M	240M	100%

Most accesses, 77%, take place for the (luminance) pixel accesses in padding, motion estimation (full-pel and half-pel), and motion compensation. The second largest contribution, 21%, is due to shape coding including the shape motion estimation. The remaining 2% is needed for the MB coding. Our methodology has been applied to the entire luminance motion part, because it is a representative and independent piece of code (functions and data). In this paper we will only focus the detailed discussion on the luminance full-pel motion estimation which takes 87% of the accesses within the optimized part.

VII. Global Loop Transformations

The goal of loop transformations is to reduce power and memory costs by optimizing the access structure. For multimedia applications it is important to apply those transformations globally over the entire algorithm. In practice, the power and memory costs cannot be accurately determined yet, in which case estimates with user-defined weights should be used. Optimizing criteria for the transformations are defined in terms of regularity and dependency length [39]. Data dependency crosses[1] are not regular and need much intermediate storage. Loop transformations like reversing a loop can solve data dependency crosses but can put the non regularity somewhere else. When there are many loops it is like a N-dimensional puzzle which has to be solved to find the optimum. A good measure for the dependency length is data locality per signal. Data locality is a measure for how close the accesses are toward each other, per signal element. If the accesses are close

[1] E.g., the first calculated element in a function is last needed in the next function and vice versa.

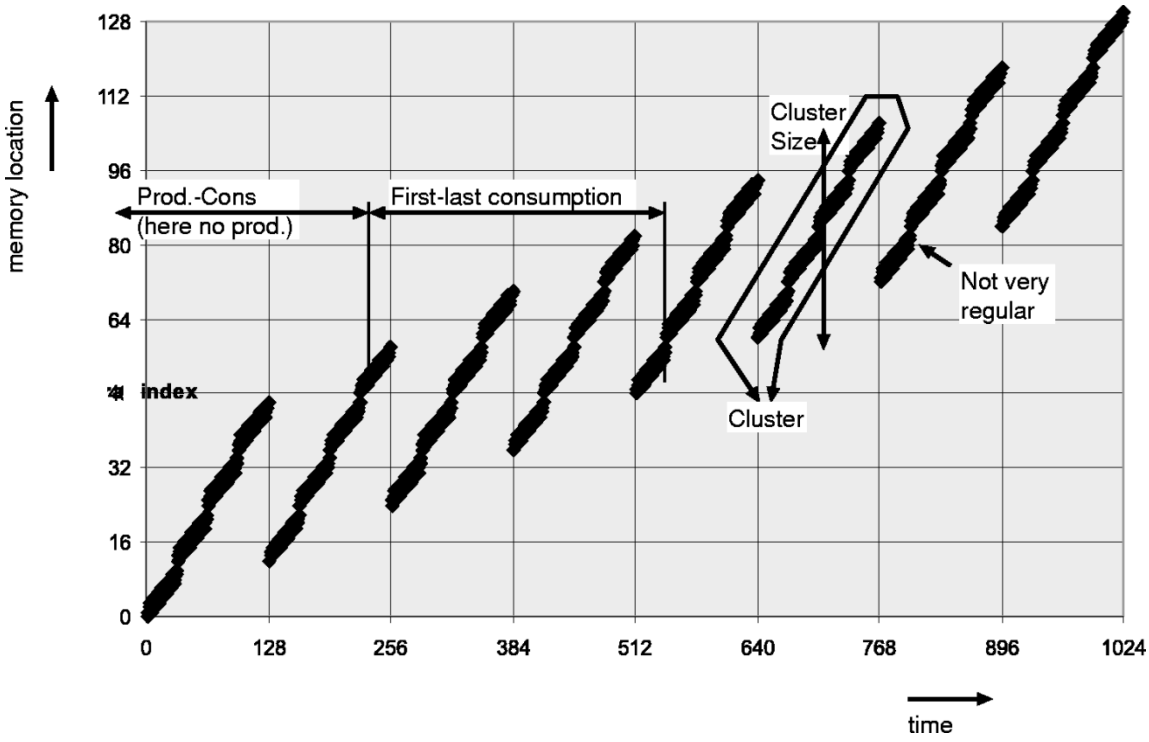


Fig. 3. Locality characteristics in access graph.

to each other, the element can be reused in foreground memory, or at least closer to the data-path (than main memory). By applying global transformations, the access structure and locality will heavily change. Shortening the global production–consumption and consumption–consumption distance is the main goal for locality improvement. Sometimes data-flow bottlenecks have to be broken to allow this and this requires also global data-flow transformations for which we also have a systematic approach [8].

A data access graph shows which signal element is accessed when in time (e.g., Fig. 3). To get a clear view on the access structure we have used a motion estimation algorithm with very small parameters. The next items judge the graph for data locality and required memory size in the stages close to the CPU datapaths.

- *Consumption near to production, especially the avoidance of high cost dependency crosses.* This is recognizable as thin graphs. The advantage is the short storage time.
- *Consumption near to consumption.* This is also recognizable as thin graphs. This will lead to a good data reuse [15].
- *Clustered reads.* This is a special case of the previous item. Multiple small bands of consumption (read and fast reread). This clustering will lead to good multiple level reuse [15].
- *Size of clusters.* The size of the clusters is an estimate of the size of an intermediate memory in the memory hierarchy.
- *Production and consumption in the same order.* To make loop merging possible.

During this entire step, signal size and (estimated) access count have to be used to weigh the importance of every signal because the improvement of signal locality for one signal usually has to be traded off with less locality for other signals. A formalized methodology to deal with such loop and data flow transformations in an embedded processor context has been proposed in our group [14], [39]. To illustrate this in the MPEG-4 context, the inner four loops of the motion estimation routines will be explored in depth. For clarity considerations, only the MB (not block) oriented motion estimation is considered in this section. In addition, the graph should look as regular as possible because then the complexity of the resulting program will be simpler.

```
   most          most      original loop structure
   outer         inner
1) y_s, x_s, y_p, x_p <-- for (y_s) y of search loop
2) y_p, x_p, y_s, x_s       for (x_s) x of search loop
3) y_s, y_p, x_s, x_p        for (y_p) y of SAD calc. loop
4) y_p, y_s, x_p, x_s         for (x_p) x of SAD calc. loop
5) y_s, y_p, x_p, x_s
6) y_p, y_s, x_s, x_p
```

Fig. 4. Loop interchange possibilities.

Loop interchange and loop folding do have a big impact on the access structure [32] of the previous VOP. To interchange the loops, $4! = 24$ possibilities exists. Due to symmetry reasons, the 24 loop interchange methods are reduced to 6 (numbered 1 to 6). Each group of four related methods will have an equal temporal data locality (not spatial locality). Only the row-major solutions have been explored. These are shown in Fig. 4.

The loop folding transformation possibilities of method 1 are illustrated in Fig. 5. The big outer square stands for the

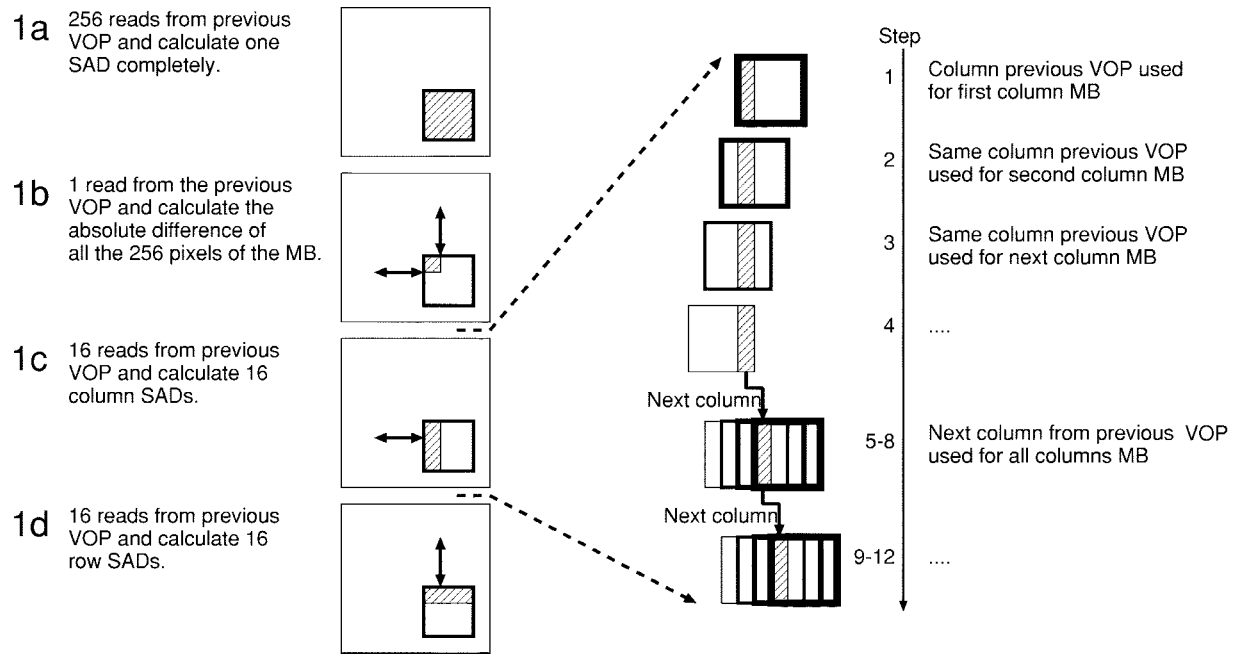


Fig. 5. Folding solutions of method 1.

search area. In the search area you will find the current MB (smaller square). The dashed area are the pixels which are read in every step in the search area. The 1a method is the original loop folding (in fact no folding), the entire MB match is read from the previous VOP.

Method 1c is expanded to the right, it shows the different steps in the SAD calculation. In the first step, the column of previous VOP pixels is used for the first column SAD of the current MB. Next, the same column of the previous VOP is used for the next column SAD of the current MB. Obviously these two SAD's do not relate to each other and are only intermediate results. This will be repeated until the last column of the current MB is reached. It is as if the MB is shifted over one column of previous VOP pixels, resulting in 16 intermediate values of a 1/16 SAD. In the next phase, the next column of the previous VOP will be read and used in the same way until all the columns in the search area are processed.

The loop folding is performed in the other direction in 1d and in both directions in 1b. In all cases the entire search area is traversed but the number of previous VOP pixels per position changes. For instance method 1b reads only one pixel from the previous VOP and "moves" the current MB over it. Some edge effects do exist (at the loop bounds), but ignoring these the transformation can be performed by changing the index calculation only, as shown in the following source codes:

```
for (y_s = 0; y_s < Y_S_MAX; y_s++)
 for (x_s = 0; x_s < X_S_MAX; x_s++)
  for (y_p = 0; y_p < Y_P_MAX; y_p++)
   for (x_p = 0; x_p < X_P_MAX; x_p++)
    sad[x_s][y_s] += abs(prev_VOP[x_s+x_p][y_s+y_p] -
                         curr_MB[x_p][y_p]);
```

Source 1: Version 1a; no loop folding

```
for (y_s = 0; y_s < Y_S_MAX; y_s++)
 for (x_s = 0; x_s < X_S_MAX; x_s++)
  for (y_p = f(y_s); y_p < g(y_s); y_p++)
   for (x_p = f(x_s); x_p < g(x_s); x_p++)
    sad[x_s-x_p][y_s-y_p] += abs(prev_VOP[x_s][y_s] -
                                curr_MB[x_p][y_p]);
```

Source 2: Version 1b; x & y loop folding

```
for (y_s = 0; y_s < Y_S_MAX; y_s++)
 for (x_s = 0; x_s < X_S_MAX; x_s++)
  for (y_p = 0; y_p < Y_P_MAX; y_p++)
   for (x_p = f(x_s); x_p < g(x_s); x_p++)
    sad[x_s-x_p][y_s] += abs(prev_VOP[x_s][y_s+y_p] -
                             curr_MB[x_p][y_p]);
```

Source 3: Version 1c; x loop folding

```
for (y_s = 0; y_s < Y_S_MAX; y_s++)
 for (x_s = 0; x_s < X_S_MAX; x_s++)
  for (y_p = f(y_s); y_p < g(y_s); y_p++)
   for (x_p = 0; x_p < X_P_MAX; x_p++)
    sad[x_s][y_s-y_p] += abs(prev_VOP[x_s+x_p][y_s] -
                             curr_MB[x_p][y_p]);
```

Source 4: Version 1d; y loop folding

Loop folding provides per interchange at least 4 possibilities (numbered a to d). From now on we refer to loop transformation by using a number followed by a character, the number for the interchange transformation (see Fig. 4), the character for the loop folding (Fig. 5). Our experiment is limited to these two most promising types of loop transformations for the motion estimation example which already exhibit 96 combinations. By performing these transformations, the data locality of the previous VOP will improve, but since the entire SAD calculation is not completed at once, the intermediate

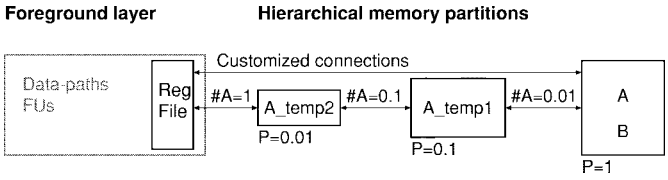


Fig. 6. Memory hierarchy.

SAD values need to be stored. By improving the data locality of one signal, you will usually get a worse locality in another signal for exchange; so a global exploration is needed to find the optimum. The exploration space has been systematically searched and reduced in size. The number of transformation candidates is then reduced from 96 to 4 (1a, 1c, 1d and 3a). Among them the original loop order 1a is still present. The selection has been done in two steps. First, by crudely checking on the data locality and a high-level estimation of the possible data reuse (see Section VIII). Second, by using a data access graph representation which gives an accurate overview of which element is accessed when.

VIII. Data Reuse Decisions

By adding intermediate memories, the data can be kept closer to the data path. This is related to the traditional caching policies, but the approach used here is much more application oriented, aggressive and global [15]. Power savings can be obtained by accessing heavily used data from small memories instead of from the large main memory. The optimization will introduce copies of the data from large to small memories. The data will be copied from the main memory to the smaller intermediate memory and accessed (read) multiple times from the smaller memory. On the one hand, power consumption is decreased because data is now read mostly from small memories, while on the other hand, power consumption is increased because extra memory transfers are introduced. Moreover, adding another layer of hierarchy can also have a negative effect on the total memory size and interconnect cost, and as a consequence also on the power [15]. So, the power per memory access will decrease and the number of accesses will increase the closer it is to the data-path (see Fig. 6).

In our multistep data reuse methodology, several issues can be decoupled, allowing the potential reuse to be explored (more extensively) per signal first. For this purpose, a structural data reuse tree is built per signal. Afterwards, merging of the final selections in the distinct data reuse trees is necessary, by assigning different intermediate memories (of different signals) to the same physical memory [15]. In this way, the number of actually needed memories can be fitted to meet external processor constraints (most processors support only two or three levels of memory hierarchy).

The important signals in the motion estimation are: SAD, alpha, curr,[2] prev. The discussion below will explain how to explore all the reuse possibilities in a structured way. The *prev* signal is the most nonlocally accessed signal. By simply looking at the motion estimation a MB level data reuse is visible. As the SAD is calculated for one position in the search area, the previous VOP pixels can be stored in a memory of the size of one MB. At the next position, most pixels can be read from the small memory for the SAD (see Fig. 7). In total $15 \times 16 = 240$ out of 256 pixels from the previous VOP can be reused.

A second look reveals the opportunity to reuse data also from the previous line. The MB is sliding over the search-area from left to right and at the end of a line the sliding MB will go over the next line one pixel lower. By exploring this reuse 255 of the 256 pixels can be reused. The disadvantage is the bigger intermediate memory needed for the reuse (1280 pixels for a band over the SA instead of only 256 pixels for storage of one MB block). The number of reads to the band is the same as to the MB cache but the reads for the bigger memory are more power hungry. Another solution adds both intermediate memories (MB band and MB block).

The same band and block reuse approach can be applied on the search area level (i.e., when the search area moves one MB).

When we look at the loop structure, we see a strong correlation between the levels of potential data reuse and the loops (see below). In the two most inner loops there is no reuse available because every previous VOP pixel is accessed only once within the scope of the two loops.

```
for y_MB(current VOP)    <--> Search area band
 for x_MB(current VOP)   <--> Search area block
  for y_s(earch_area)    <--> MB band
   for x_s(earch_area)   <--> MB block
    for y_p(ixels_of_MB)
     for x_p(ixels_of_MB)
```

Different levels of reuse can be combined to obtain the optimum signal hierarchy. It is obvious that the number of solutions is huge when all possibilities are searched. Fig. 8 shows the data reuse trees for method 1a. This is a structured way of showing all the reuse possibilities. The most upper solution in the tree (root) is the original solution (dissipating nearly 30 W for the *prev*ious VOP signal), nothing is reused and everything is read from main memory. It is clear that this is not acceptable and a memory hierarchy is needed.

All the nodes in the tree of Fig. 8 offer a valid solution for reuse. In the first level of hierarchy all the four types of reuse are considered: MB block, MB band, SA block or SA band. When we add a second level, then the first level exploration remains valid and every solution is again subdivided in multiple next level solutions. It only makes sense if the intermediate memories become smaller as they come closer to the data-path (so a limited number of solutions remains).

The estimated size of the intermediate memories and a high level estimation of the number of reads and writes are fed into a memory power model [38], [10] to estimate the power for every reuse strategy. The power numbers, in Table II, are corresponding to the numbers in the figure and include the power of all the levels of the memory hierarchy of the branch. Our experiments have shown that by changing the power

[2] We will use the worst case since there are no statistics available; all pixels inside the shape → alpha and curr will be equal.

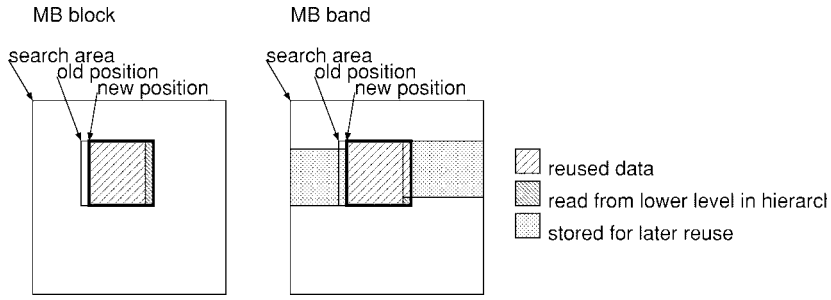


Fig. 7. Data reuse block and band.

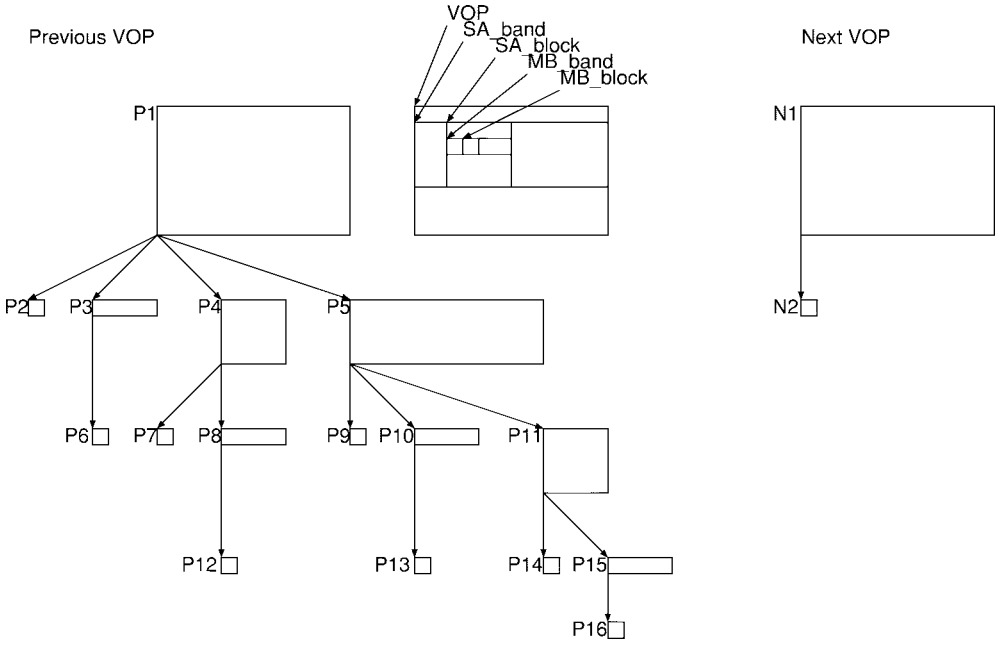


Fig. 8. Reuse trees for code version 1a. Left: reuse tree for previous VOP. Middle: how do these intermediate memories match each other. Right: reuse tree for next VOP.

TABLE II
NUMBER OF READS FOR THE THREE PARTITIONS

Previous VOP				Next VOP	
Reuse	P in mW	Reuse	P in mW	reuse	P in mW
P1	29949	P9	862	N1	27805
P2	2783	P10	685	N2	601
P3	827	P11	874		
P4	898	P12	737		
P5	2785	P13	719		
P6	861	P14	719		
P7	743	P15	**679**		
P8	703	P16	712		

model these result can change significantly. So the memory model and the memory hierarchy (if it is fixed) need to be given to find the optimum. However, the same methodology can be used for any other power model.

The data reuse trees for the four different transformed methods are built in a similar way. The tree structure is equal due to the fact that the number of loops hasn't changed, but they have different power figures. Moreover, they require different branches in the data reuse tree for the power optimal decision. A large total gain of a factor 46 in memory power is reached (compared to the reference code) by choosing the optimal transformation and branch in the reuse tree. Because of implementation ease, the original loop ordering with a custom data reuse solution is chosen for further exploration. The loss due to this nonoptimal choice is 1.4% only (from 1856 mW to 1882 mW).

Power can be traded off for memory size and hence chip or board area. Fig. 9 shows the total memory size put against the needed power. Implementation 15 is best for power, but this will need a huge amount of intermediate memory which is also three levels deep. Good alternatives for power and area are: 3, 7 and 8. By choosing number 7, the memory hierarchy size is reduced from 32 kB (optimum branch) to 6.4 kB by only sacrificing 3.3% of the total memory power (from 1882 mW to 1945 mW). These two levels also do fit best onto the TriMedia processor memory architecture so they have been retained for the final solution.

IX. COMBINING MOTION ESTIMATIONS

All the experiments below have been performed using the real code and the access counts are based on actual profiling for real VOP streams.

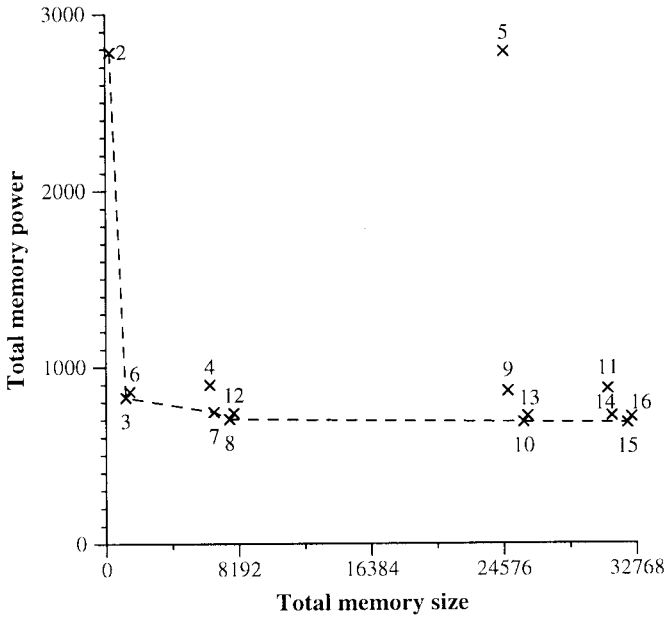


Fig. 9. Memory size versus power for prev signal.

A. Merging Possibilities

Up to now, only a simplified motion estimation kernel has been analyzed and discussed. Now, the entire motion estimation of a group of VOP's will be studied and globally optimized. Source 5 reflects the original motion estimation code of the MPEG-4 encoder [33] (see also Section II). First the entire P-VOP is coded, followed by the successive B-VOP's. Because of this implementation, the search area window will go M (=number of VOP's in a group) times through the previous VOP and $(M-1)$ times through the next VOP. By combining the different motion estimations which take place on the same VOP and position, a significant gain can potentially be achieved (up to a factor M in reads).

```
for (all MBs P-VOP)
  for (all position in search area)
    for (all pixels MB)
      sad += abs(prev_vop[][] - next_vop_MB[][])
for (all B-VOPs of group; M-1 times)
  for (all MBs B-VOP)
    for (all position in search area on prev VOP)
      for (all pixels MB)
        sad += abs(prev_vop[][] - curr_B_vop_MB[][])
    for (all position in search area on next VOP)
      for (all pixels MB)
        sad += abs(next_vop[][] - curr_B_vopMB[][])
```

Source 5: Pseudo code for MPEG-4 motion estimation of a group of VOPs

However, not all motion estimation steps can be merged (infeasible due to data dependencies). To make the differences clear, the following naming convention is used for the different motion estimations in the studied alternatives. The "P" motion estimation is from the next P-VOP to the previous P-VOP. The abbreviation "Bn" (or "Bp") is used for the all B-VOP (of one group) motion estimations in one direction, the n for Next, the p for Previous.

The original order is displayed in Fig. 10 as V1, with no merging at all. Version V2 and V3 use the VOP's which were originally stored in the frame reordering hardware. When incorporating the hardware (memory) needed for the reordering into the coder, these alternatives will not negatively affect the total amount of memory to store VOP's or delay/latency for the rest of the system. The other two versions (V4 and V5) need twice as many VOP's to store and the latency will at least double, which is not acceptable.

Version V2 executes the motion estimation for all MB's of the P-VOP first and when finished, the motion estimation of the B-VOP's in both directions are executed in parallel (see also Source 6). The search area traverses the previous VOP twice per group (for P and for Bp) and the next VOP once (for Bn). During the B-VOP motion estimation, two search areas are active concurrently.

```
for (all MBs P-VOP)
  for (all position in search area)
    for (all pixels MB)
      sad += abs(prev_vop[][] - next_vop_MB[][])
for (all MBs B-VOP)
  for (all B-VOPs of group; M-1 times)
    for (all position in search area on prev VOP)
      for (all pixels MB)
        sad += abs(prev_vop[][] - curr_B_vop_MB[][])
    for (all position in search area on next VOP)
      for (all pixels MB)
        sad += abs(next_vop[][] - curr_B_vopMB[][])
```

Source 6: Pseudo code for version V2

In version V3, another merging is used (see also Source 7), where *all* the motion estimations on the previous VOP (P and Bp) are combined and afterwards *all* the motion estimations on the next VOP (Bn) are combined. Hereby, version V3 has two traversing search areas only (less accesses to fill two search areas instead of the three of V2) which are not concurrently needed (smaller intermediate storage due to better inter signal inplace [13]; the two search areas can use the same memory after each other). Both characteristics (accesses and size) advocate for version V3. But before coding the MB, both best matches of the motion estimation (from previous and next VOP) must be read for the interpolated mode. This means that 256 pixels must be read from the previous VOP and 256 pixels must be read from the next VOP.[3] For data locality reasons[4], the interpolated mode and MB coding is executed directly after the motion estimation on the next VOP. Also in this way, the best match from the next VOP must still be in the next VOP search area memory since the search hasn't moved and the best match has just been found in this search area. But the previous VOP search area is not available anymore (since the previous VOP best match was determined in the other loop nest, see Source 7), so these reads have to come from the VOP (main) memory. This is unlike version V2, where the previous and next VOP are available at the same time.

[3]The direct mode reads need to be taken into account as well, but for complexity reasons we will not add these results here.

[4]The current MB does not have to be reread then.

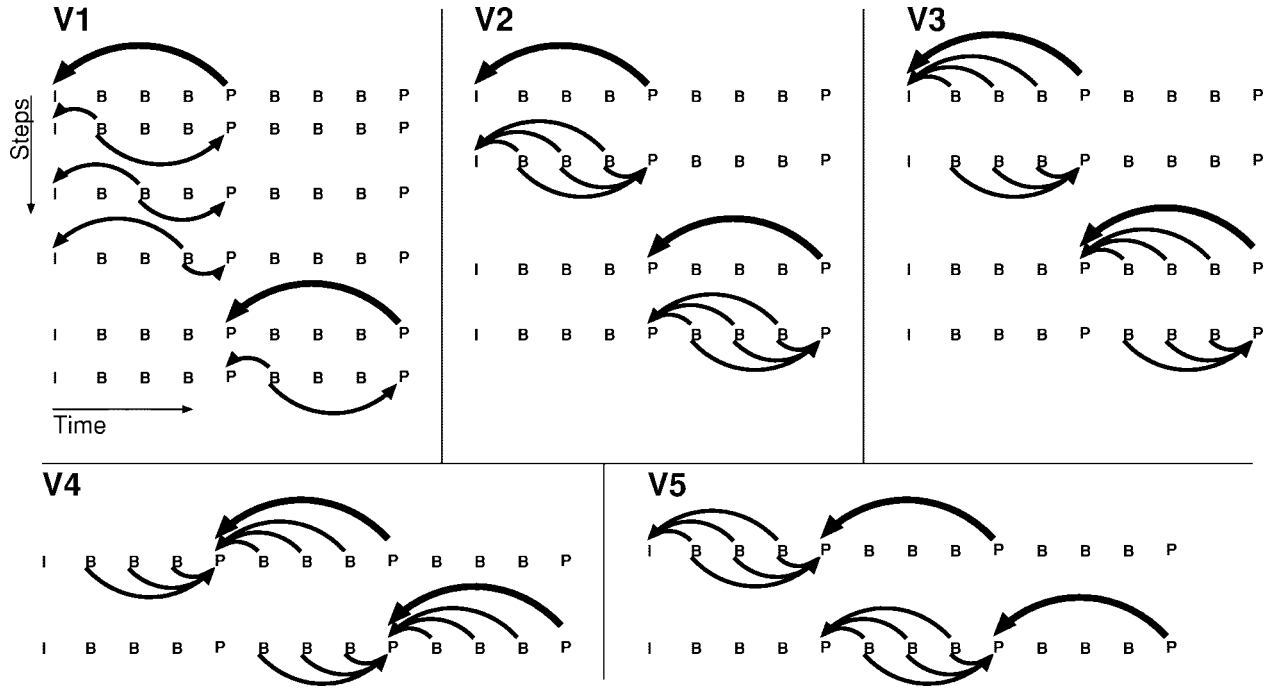


Fig. 10. Motion estimation merging possibilities with data dependencies between VOP's.

```
for (all MBs)
  for (all position in search area)
    for (all pixels MB)
      sad += abs(prev_vop[][] - next_vop_MB[][])
  for (all B-VOPs of group; M-1 times)
    for (all position in search area on prev VOP)
      for (all pixels MB)
        sad += abs(prev_vop[][] - curr_B_vop_MB[][])

for (all B-VOPs of group; M-1 times)
  for (all MBs B-VOP)
    for (all position in search area on next VOP)
      for (all pixels MB)
        sad += abs(next_vop[][] - curr_B_vopMB[][])
```

Source 7: Pseudo code for version V3

Without going in too much detail, we show the increased complexity of the MPEG-4 context. In the "static" case, for MPEG-2 and H.263 coders or alike, the MB's having an equal position in the frame would be combined because they have an equal search area. However, in MPEG-4 the VOP's do not (have to) coincide, so the MB's and their search areas will not necessarily overlap fully. Combining the MB's on the same position within the bounding box may cause a poor or even no overlap between the corresponding search areas. This results in a poor reuse and moreover, the combined search area has no upper size limit (which is hard to optimize at compile time). By defining a maximum distance between the MB's, both problems will be solved; there will be an overlap because the MB's are not too far from each other and the combined search area will maximally grow the maximum distance. The maximum distance for which the merging is performed is a tradeoff. By making it small it will combine the motion estimations poorly. On the other hand, a large distance will need a big combined search area. Experiments, worst case calculations and implementation complexity estimations have pointed out a maximum distance of 16 (=MB_SIZE) as a good compromise. This way, every VOP will contribute one MB to every combined motion estimation (if one inside the bounding box). It cannot contribute two MB's because then the maximum distance is exceeded and there will be at least one inside the 16×16 area.

B. Main Memory Read Count Estimation

A similar reuse decision exploration is used as in Section VIII, but the power figures have changed. To make a fair tradeoff between V2 and V3, we estimate the number of main memory reads. The absolute number of reads depends on VOP size, shape, group size, etc. At compile time we don't know size, shape nor group size. However, we can make a relative comparison for the number of accesses for a group of VOP's. Version V2 has three search areas and V3 has only two. They will have an equal relation in the number of accesses. The additional reads for the interpolated mode are needed for a good estimation. The number of reads depends on VOP size for which we will assume the CIF format here. Moreover, the group size is important as all the members of the combined motion estimation will read one MB from the previous VOP, so the number of reads will grow linearly with the group size.

Table III and Fig. 11 give the estimated number of VOP reads and the needed total search area memory size. One search area can maximally grow to 64×64 which is 4 kB, so V2 needs 4 kB and V3 needs 8 kB. The number of main memory reads for a search area to traverse a VOP equals:[5]

$$\#R = W_{\text{prev\_VOP}} \times H_{\text{SA}} \times nr\_MB\_\text{rows}_{\text{next\_VOP}}$$

[5] Assuming reuse when the search area traverses in horizontal direction.

TABLE III
MEMORY SIZE AND MAIN MEMORY ACCESSES FOR V2 AND V3

	M	#R main memory (×1000)	average *r*eads	SA mem size (#byte)
V2	var	1216		8192
V3	var	884 + 230×(M-1)		4096
V2	8	1216	9.9	8192
V3	8	2494	20.3	4096
V2	4	1216	9.9	8192
V3	4	1574	12.8	4096

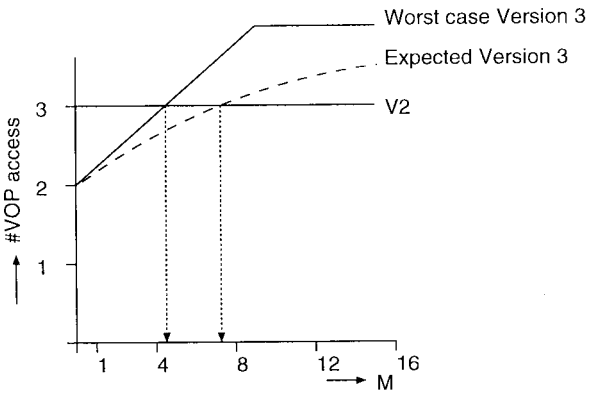


Fig. 11. Number of accesses for V2 and V3.

For every row of MB's in the next VOP ($nr\_MB\_\text{rows}_{\text{next\_VOP}}$), it will have to read a band of pixels which has the previous VOP width[6] ($W_{\text{prev\_VOP}}$) and the search area height (H_{SA}). For example the P motion estimation (like in phase 1 of version V2) needs 332 k reads (=384 × 48 × 18). However, a Bp (and Bn) motion estimation requires a bigger search area (due to the VOP offset), so it needs 442 k reads (=384 × 64 × 18). The total number of reads for version V2 is 332+442+442 (P + Bp + Bn).

For version V3, an access count is needed for the previous VOP reads in the interpolated mode, direct mode and motion compensation. Per B-VOP MB 580 reads are needed from the previous VOP (assuming already an optimized implementation for the different modes). Thus, the combined B-VOP MB requires 580 × 22 × 18 × (M-1) reads. The total number of reads for version V3 is 442+442+230 × (M-1) (Bp + Bn + modes).

The break-even point between V2 and V3 is at $M = 1,44$. This means that you will always choose for V2.[7] This is because we assumed a worst case read count for the interpolated mode reads. A clever implementation will take advantage of the overlap for the interpolated mode reads. Experiments have pointed out that the motion vectors of the MB's which are combined all point in the same direction and the needed pixels do have a big overlap. This can easily be explained due to the inertia of moving objects. The bigger the group becomes, the bigger the probability of a overlap and the less extra reads are needed (see Fig. 11). Also, the number of reads is limited by the search area size. The break-even point will shift to a higher M.

[6] Including padded edge.

[7] $M = 1$ is not a option because then there are no B-VOP's. In that case there is no difference between V2 and V3.

C. Selection of Optimal Implementation

Multiple implementations are possible to build one of the two different versions. This section gives the opportunity to select a good alternative for a given memory (or cache) size. The end of Section IX.A explains some of the remaining implementation freedom. Without going into more detail, Fig. 12 shows the number of memory accesses versus the required memory for different implementations (both several version V2 and V3). Clearly, most implementations are not useful since they perform worse in both (accesses and memory size). The dotted lines connects the interesting implementations. The most suitable implementation for a given memory size (which is known for every MM-processor) can be found by reading this curve. Here, in the case of TriMedia which has 8 K internal memory, we've chosen for version V2a. Version V2h is slightly better for power but is much more complex to implement.

Moreover version V3a is build to verify the estimations of Section IX-B. Fig. 13(a) shows the measured result for the first 96 frames of the "hal" sequence. The break-even point has moved to $M = 10$. In the original source, the number of reads from the (current) B-VOP's was negligibly small. Now after optimizing, the analyzed number of accesses from the current VOP has a larger share. An important observation here is that the (current) B-VOP reads for V3 increases twice as fast as V2 [see Fig. 13(b)]. This is caused by the separation of the loops: for the motion estimation on the previous and next VOP, all the B-VOP's are read twice. Adding these reads moves the breakpoint close to $M = 6$, in the middle of the most common values of M (4 and 8).

X. OVERALL RESULTS FOR MPEG-4

We have applied our DTSE methodology on the MPEG-4 motion estimation and have analyzed the background memory gain (the entire memory hierarchy without foreground registers). A measured gain of a factor 65 in luminance pixel VOP memory accesses is obtained by simulation of the first 96 frames of the reference "hal" sequence in CIF resolution. The search area memory power is reduced with a factor 5.3. The MB oriented memory accesses is increased with 50% but these involves small signals only. The total background memory power gain is a factor 8.0 (see Fig. 14) which will make other parts of the processor power dominant. In the current optimization we have mainly focussed on the VOP and search area memory. If necessary, we expect to be able to undo the power increase in the MB oriented memory and to have an overall memory power gain of a factor 10 to 15.

XI. CACHE BEHAVIOR OF A TRANSFORMED H.263

In this section, the cache hit rates of a software implementation of H.263 video decoder will be determined. Then it will be shown that applying our custom hardware oriented data transfer and storage optimization methodology, which mainly

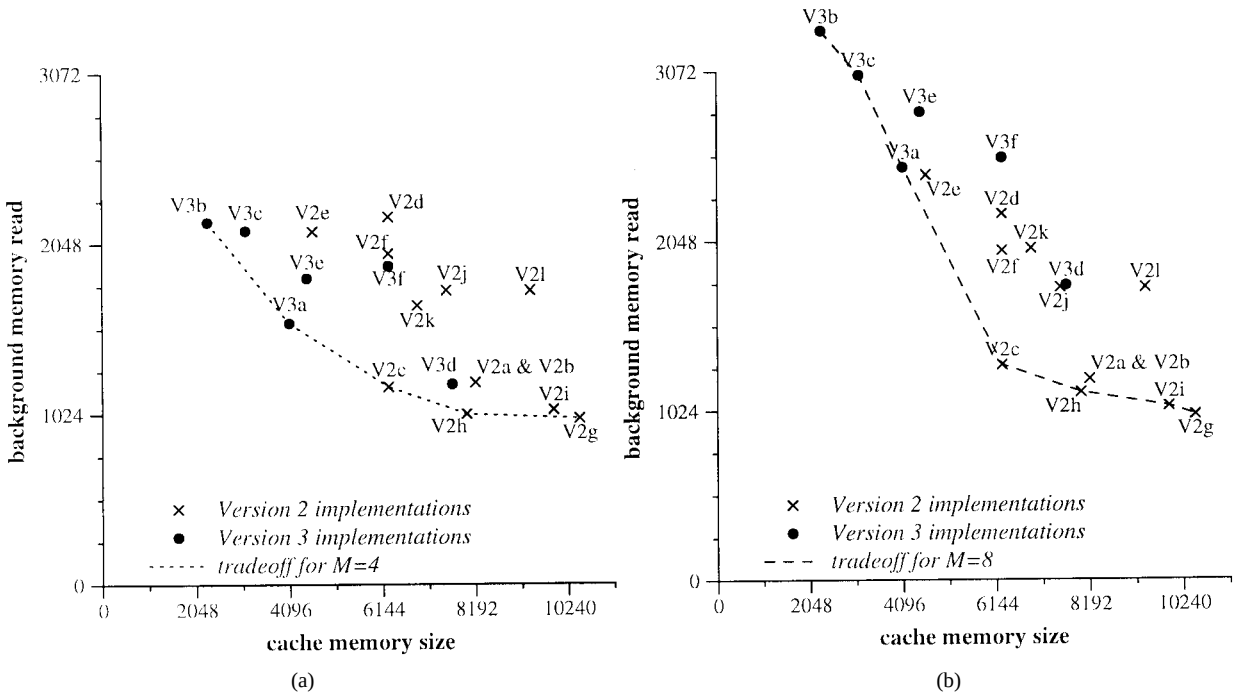


Fig. 12. Memory requirement versus memory reads.

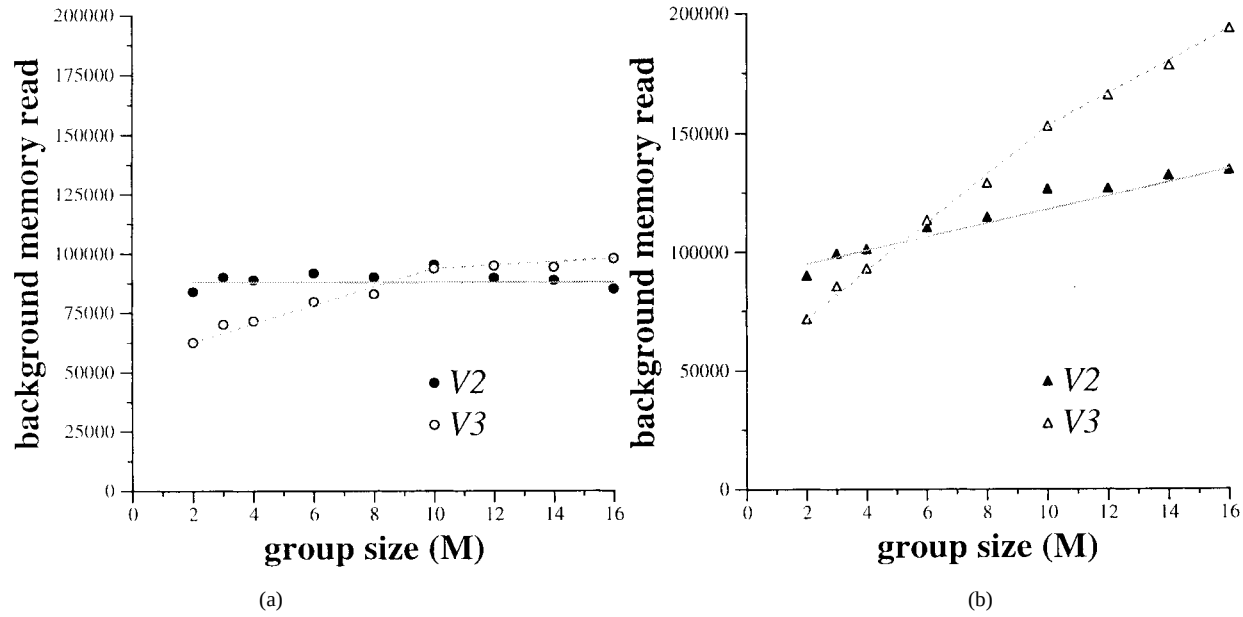


Fig. 13. Measured accesses for V2 and V3. (a) Access count from previous and next VOP; (b) also includes the reads from the B-VOP's.

focuses on reducing the power consumption and memory size reduction [9], also improves the cache hit rates and lowers the number of cycles spent to access the primary cache of processors like the DEC Alpha.

Starting from the public domain code for a H.263 video conferencing decoder [11], the number of transfers to arrays present in the code are counted while decoding a H.263 video stream called `suz14.263`. The stream decodes to 75 Quarter CIF (QCIF) frames which corresponds to 2.5 seconds real-time video for a frame rate of 30 frames/s. One QCIF frame is 176 pixels wide and 144 pixels high. Since the chrominance values are subsampled, a total of $176 \times 144 + 2 \times 76 \times 88 = 38\,016$ bytes are needed to store 1 frame. The length of the encoded stream depends on the modes enabled during encoding. The main data flow of this video decoder in the initial code description is depicted in Fig. 15.

The video decoder was simulated assuming the memory organization tabulated in the column "Original" of Table V. The organization of the memory corresponds to what a traditional C compiler typically does. It places arrays one after each other in the same order as the declaration.

In our experiment, the cache system is configured like the DEC AXP 3000/800S. This processor has a direct mapped, write through primary cache of 8 kB, cache lines of 32 bytes

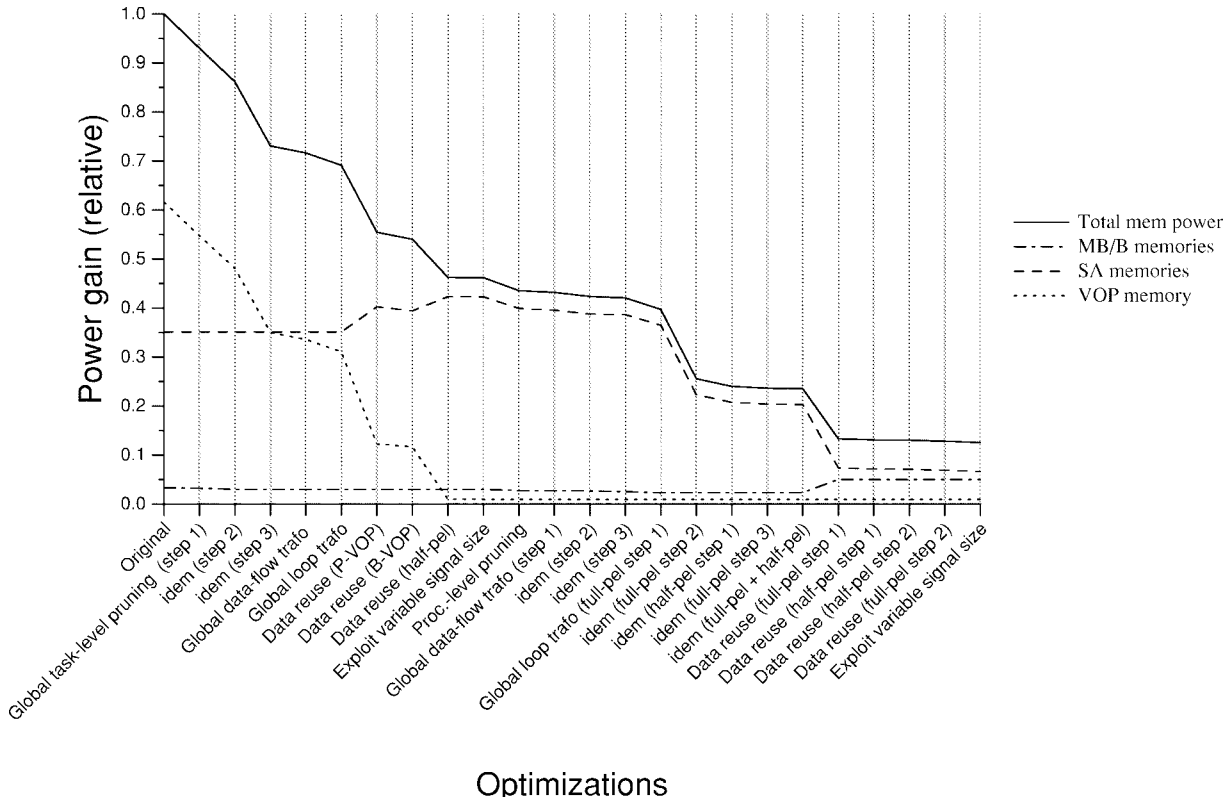


Fig. 14. Memory power gain per step.

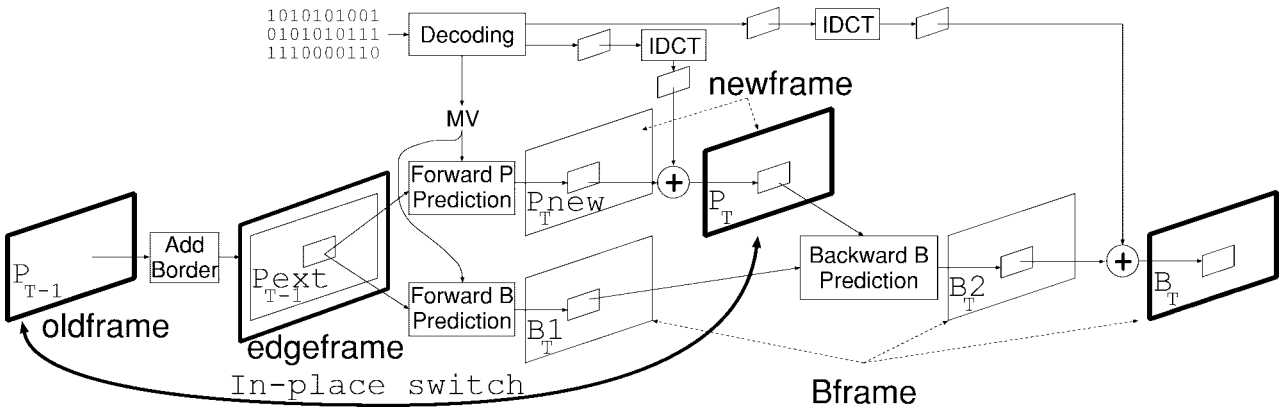


Fig. 15. Data flow and main tasks in the reference H.263 video decoder.

[4] and a write allocate upon a cache write miss. Because the total memory size is 170 774 bytes, we may assume that all data is cached by the secondary cache that is at least 256 kB large. The number of compulsory misses[8] is relatively low compared to the total number of accesses from the L2 cache and can be ignored.

The number of cache hits and misses are obtained by an accurate simulation of the cache using C++ classes. The result of such a simulation for the video decoder software implementation of Telenor Research [11] is presented in the second and third column of Table IV.

To estimate the number of cycles due to reading and writing to the caches, we assume that one processor cycle is spent upon a L1 cache hit and 8 cycles if the L1 cache is missed. Hence the total number of cycles due to transfers to/from the L1 cache is the total number of hits plus eight times the number of L1 cache misses. This is a best-case estimation and in practice it will be even worse (since the compulsory L2 cache misses are not taken into account and not every processor cycle will contain a data access).

The hit rates of the cache can be improved by applying some of our data storage and transfer optimizations on the software implementation of Telenor Research. For an explanation of the full methodology for custom memory organizations, we refer to [9], [28], [29]. We briefly list the major optimizations that were performed to obtain a power efficient implementation.

- Global data-flow transformation: removal of all accesses related to the border.
- Global loop and control flow transformations: e.g., merging of the forward and backward predictions.

[8] Cache misses which cannot be avoided because the data is read for the first time.

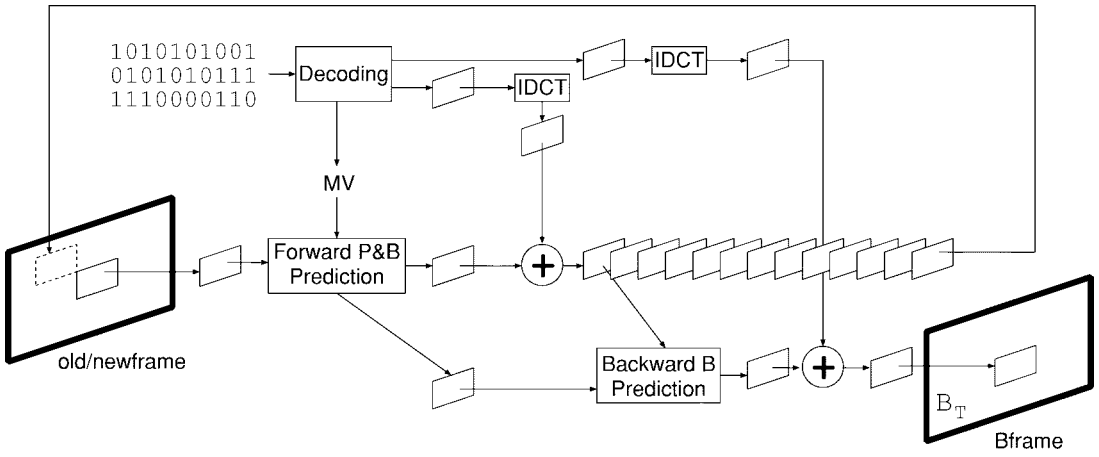


Fig. 16. Optimized H.263 decoder.

TABLE IV
PRIMARY CACHE PERFORMANCE OF THE DEC AXP 3000/800S WHILE DECODING WITH THE ORIGINAL AND OPTIMIZED DECODER

	DEC AXP 3000/800S	
Number of	Original	Optimized
cache hits read	17,146,377 (91%)	22,376,986 (96%)
cache miss read	1,695,701 (9%)	716,834 (4%)
tot. read cycles	30,711,985	28,111,658
cache hits write	7,803,255 (84%)	13,003,941 (94%)
cache miss write	1,461,809 (16%)	721,095 (6%)
tot. write cycl.	19,497,727	18,772,701
tot. tr. cycles	50,209,712	46,884,359
cache reads	18,842,078	23,093,820
cache writes	110,305,384	59,738,764
L2 cache reads	101,040,320	46,013,728
L2 cache writes	9,265,064	13,725,036

TABLE V
MEMORY ORGANIZATION OF THE ORIGINAL. AND OPT. VIDEO DECODER

	Nr. of bytes	
Content	Original	Optimized
motion vectors	1300	1300
mode of each MB	130	130
blocks for the OBMC	64	64
OBMC constants	320	320
`old/newframe`	2×38016	38016
`edgeframe`	54912	0
Bidir. predicted frame	38016	38016
forward predicted MB	0	384
forward OBMC block	0	64
backward predicted MB	0	384
buffer of 13 MB's	0	4992
Total	170774	83670 (48%)

- Memory hierarchy exploitation: introduction of small local buffers and the corresponding data transfers, to buffer the forward predicted MB, the bidirectional interpolated MB, several tables like `OM`, `MV` (motion vectors) and `modemap`, a MB to store the result of IDCT. For the macro block related data, 3 extra levels of storage hierarchy were introduced in the most optimized solution to fully exploit all data reuse.
- In-place optimizations: reduction of `old frame` and `new frame` to 1 frame and an in-place buffer of 13 MacroBlocks.

These optimizations reduce the accesses to frame memories drastically by making use of local distributed memory hierarchy. The data flow between the distributed small local memories and the frame memories of the improved video decoder is depicted in Fig. 16. This new code and the custom memory organization lead to an overall power saving in the (dominant) memory network of a factor 9 in maximum power for the worst-case OBMC mode [28], [29].

Although the optimizations mentioned above are carefully selected with an application specific memory organization in mind, the resulting description can also be used when targeting a software implementation on a general purpose processor. It's interesting to measure the impact on the cycle count when a power optimized description is mapped onto general purpose processors, because many designers would probably claim that overhead to introduce power savings would have a negative effect on the performance. We will show however that this is not true due to the aggressively improved data access locality in the code. So significant power reduction can be obtained without sacrificing on overall speed.

The improved video decoder holds the data tabulated under "Optimized" in Table V.

The total size of the memory is 83 670 bytes. This is 48% of the number of bytes needed to store the data structures of the reference code.

The cache behavior for this optimized decoder is also simulated. The results are tabulated under "Optimized" in Table IV.

We see that the cache miss rates decrease from 9% to 4% and from 16% to 6% for the reads and writes respectively. The cycle budget due to transfers to the L1 cache is 7% less than for the reference case.

There are 5 637 339 reads and 3 753 088 writes to the extra allocated MB's to store the result of (P and B) forward

prediction. The MB's are relatively small. For example, the luminance part of a MB is represented by four blocks of 8 × 8 bytes. One subblock of 8 × 8 pixels of 8 bit can be stored in eight general purpose registers of 64 bit. In total 32 64-bit registers are available in the DEC APX3000/800S. When those small arrays are in registers, the number of transfers to and from the primary cache will decrease even further, affecting the cycle count positively. This will also lower the power consumption of the caches.

The applied optimizations for the H.263 application were for a custom memory architecture. However, just mapping the code on a processor as is, still shows an improvement. It shows a decrease of the number of cache misses. This can be explained due to the improved data locality, and more data can retain in the cache. Redoing the design and targeting a general purpose memory structure (L1 cache—L2 cache—main memory) from the beginning would of course be better.

Since the MPEG-4 encoder has been optimized for data transfer and storage with a TriMedia memory architecture in mind, a large gain in the number of cache misses is accomplished even when we don't lock the cache. The explicit copies from the VOP to the search area memories will cause extra overhead. Also many modulo operations are used to achieve this goal. Moreover, the datatypes were not fully refined yet and no subword parallelism is used. However, no performance decrease is measured by running the code as is on the actual TriMedia board. Especially by removing the created addressing bottleneck and adding cache locking (to assure the cache hits), large performance gains can be expected also.

XII. Conclusion

Upcoming multimedia processing tasks will require high bandwidth. In this paper, we have estimated that a software implementation of an MPEG-4 video encoder (VM7.0) typically requires over 800 k main memory accesses per nontransparent MB, which is equal to 5 Gtransfers/s to encode the simple simple profile level L2.

The exploration space to optimize data storage and transfers is very large, and our experiments clearly show the importance of a formalized methodology to traverse it. We have applied our data transfer and storage exploration methodology on the MPEG-4 motion estimation and have analyzed the background memory gain (the entire memory hierarchy without foreground registers). Large gains in power, memory transfers and memory storage are achieved by building an optimized MPEG-4.

We have also shown that the cache miss rates of a heavily memory optimized implementation of a H.263 decoder decreases on a DEC "Alpha" AXP 3000/800S. So the achieved power savings can be obtained without a performance penalty, actually even combined with a relevant cycle count reduction compared to conventional reference code.

References

[1] R. Allen and K. Kennedy, "Vector register allocation," *IEEE Trans. Comput.*, vol. 41, pp. 1290–1316, Oct. 1992.

[2] S. Amarasinghe, J. Anderson, M. Lam, and C. Tseng, "The SUIF compiler for scalable parallel machines," in *Proc. 7th SIAM Conf. Parallel Proc. Scientific Comp.*, 1995.

[3] O. Arregi, C. Rodriguez, and A. Ibarra, "Evaluation of the optimal strategy for managing the register file," *Microprocess. Microprogramm.*, no. 30, pp. 143–150, 1990.

[4] P. Baglietto, M. Maresca, M. Migliardi, and N. Zingirian, "Image processing on high-performance risc systems, *Proc. IEEE*, vol. 84, pp. 917–930, July 1996.

[5] F. Bodin, W. Jalby, D. Windheiser, and C. Eisenbeis, "A quantitative algorithm for data locality optimization," Tech. Rep., IRISA/INRIA, Rennes, France, 1992.

[6] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. De Man, "Global communication and memory optimizing transformations for low power signal processing systems," *VLSI Signal Processing VII*, J. Rabaey, P. Chau, and J. Eldon, Eds. New York: IEEE Press, 1994, pp. 178–187.

[7] F. Catthoor, M. Janssen, L. Nachtergaele, and H. De Man, "System-level data-flow transformation exploration and power-area trade-offs demonstrated on video codecs," *J. VLSI Signal Process.*, special issue on System Level Trade-off Analysis in Signal Processing, vol. 18, no. 1, pp. 39–50, Jan. 1998.

[8] ———, "System-level data-flow transformation exploration and power-area trade-offs demonstrated on video codecs," *J. VLSI Signal Process.*, special issue on Systematic Trade-Off Analysis in Signal Processing Systems Design, vol. 18, no. 1, pp. 39–50, 1998,

[9] F. Catthoor, S. Wuytack, E. De Greef, F. Franssen, L. Nachtergaele, and H. De Man, "System-level transformations for low power data transfer and storage," in paper collection on *Low Power CMOS Design*, A. Chandrakasan and R. Brodersen, Eds. New York: IEEE Press, 1998, pp. 609–618.

[10] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecapelle, *Custom Memory Management Methodology—Exploration of Memory Organization for Embedded Multimedia System Design*. Boston, MA: Kluwer, 1998.

[11] Digital Video Coding at Telenor R&D, Telenor's h.263 software, version 1.3, Feb. 1995, `http://www.nta.no/brukere/DVC/h263_software/`.

[12] F. Catthoor, "Energy-delay efficient data storage and transfer architectures: Circuit technology versus design methodology solutions," in *Proc. DATE'98*, Feb. 23–25, 1998.

[13] E. De Greef, F. Catthoor, and H. De Man, "Array placement for storage size reduction in embedded multimedia systems," in *Proc. IEEE Int. Conf. Application Specific Systems, Architectures and Processors*, Zurich, Switzerland, July 1997, pp. 66–75.

[14] ———, "Program transformation strategies for reduced power and memory size in pseudo-regular multimedia applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, pp. 719–733, Oct. 1998.

[15] J. P. Diguet, S. Wuytack, F. Catthoor, and H. De Man, "Formalized methodology for data reuse exploration in hierarchical memory mappings," in *Proc. IEEE Int. Symp. Low Power Design*, Monterey, CA, pp. 30–35, Aug. 1997.

[16] J. Z. Fang and M. Lu, "An iteration partition approach for cache or local memory thrashing on parallel processing," *IEEE Trans. Comput.*, vol. C-42, pp. 529–546, May 1993.

[17] A. Faruque and D. Fong, "Performance analysis through memory of a proposed parallel architecture for the efficient use of memory in image processing applications," in *Proc. SPIE'91, Vis. Commun. Image Process.*, Boston, MA, Oct. 1991, pp. 865–877.

[18] P. Feautrier, "Compiling for massively parallel architectures: A perspective," in *Algorithms and Parallel VLSI Architectures III*, M. Moonen and F. Catthoor, Eds. Amsterdam, The Netherlands: Elsevier, 1995, pp. 259–270.

[19] K. Itoh, K. Sasaki, and Y. Nakagome, "Trends in low-power RAM circuit technologies," *Proc. IEEE*, special issue on "Low power design," vol. 83, pp. 524–543, Apr. 1995.

[20] M. Jimenez, J. Llaberia, A. Fernandez, and E. Morancho, "A unified transformation technique for multi-level blocking" in *Proc. EuroPar Conf.*, Aug. 1996.

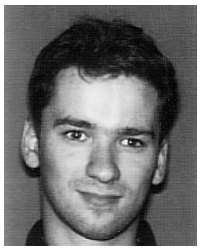
[21] R. Kleihorst, A. van der Werf, F. Bruls, W. Verhaegh, and E. Waterlander, "MPEG2 video encoding in consumer electronics," *J. VLSI Signal Process.*, vol. 17, nos. 2/3, pp. 241–252, Nov. 1997.

[22] D. Kulkarni, M. Stumm, and R. Unrau, "Implementing flexible computation rules with subexpression-level loop transformations," Tech. Rep., Comp. Systems Res. Inst., Univ. Toronto, Toronto, Ont., Canada, 1995.

[23] W. Li and K. Pingali, "A singular loop transformation framework based on nonsingular matrices," in *Proc. 5th Annu. Workshop Lang. Comput. Par.*, New Haven, CT, Aug. 1992.

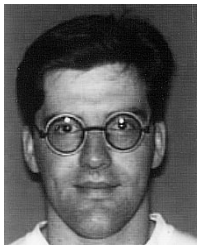
[24] P. Lippens, J. van Meerbergen, W. Verhaegh, and A. van der Werf, "Allocation of multiport memories for hierarchical data streams," in *Proc. IEEE Int. Conf. Comp. Aided Design*, Santa Clara, CA, Nov. 1993.

[25] N. Manjiakian and T. Abdelrahman, "Reduction of cache conflicts in loop nests," Tech. Rep. CSRI-318, Comput. Syst. Res. Inst., Univ. Toronto, Toronto Ont., Canada, Mar. 1995.
[26] M. Mizuno et al., "A 1.5 W single-chip MPEG2 MP@ML encoder with low-power motion-estimation and clocking," in *Proc. IEEE Int. Solid-State Circuits Conf.*, San Francisco, CA, Feb. 1997, pp. 256–257.
[27] L. Nachtergaele, F. Catthoor, B. Kapoor, D. Moolenaar, and S. Janssens, "Low power storage exploration for H.263 video decoder," presented at IEEE Workshop VLSI Signal Processing, Monterey, CA, Oct. 1996.
[28] L. Nachtergaele, F. Catthoor, B. Kapoor, S. Janssens, and D. Moolenaar, "Low power data transfer and storage exploration for h.263 video decoder system," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 120–129, Jan. 1998.
[29] L. Nachtergaele, D. Moolenaar, B. Vanhoof, F. Catthoor, and H. De Man, "System-level power optimization of video codecs on embedded cores: A systematic approach," special issue "Future directions in the design and implementation of DSP systems," *J. VLSI Signal Process.*, vol. 18, no. 2, pp. 89–109, Feb. 1998.
[30] Video group, "Text of MPEG-4 Video VM-version 7.0," N1642, Bristol, U.K., Apr. 1997.
[31] MPEG Requirement Subgroup, "Overview of MPEG-4 profiles and levels," N2325, Dublin, Ireland, July 1998.
[32] D. A. Padua and M. J. Wolfe, "Advanced compiler optimizations for supercomputers," *Commun. ACM*, vol. 29, no. 12, pp. 1184–1201, 1986.
[33] T. Sikora, "The MPEG-4 video standard verification model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 19–31, Feb. 1997.
[34] L. Stok and J. Jess, "Foreground memory management in data path synthesis" *Int. J. Circuit Theory Applicat.*, vol. 20, pp. 235–255, 1992.
[35] L. Terman and R.-H.Yan, *Proc. IEEE*, special issue on "Low power electronics," vol. 83, pp. 495–700, Apr. 1995.
[36] E. Torrie, M. Martonosi, C.-W. Tseng, and M. Hall, "Characterizing the memory behavior of compiler-parallelized applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, pp. 1224–1236, Dec. 1996.
[37] G. Slavenburt et al., "TriMedia, TM1000 preliminary data book," *Philips electronics North America Corporation*, TriMedia product group, 811 E. Arques Avenue, Sunnyvale, CA, 1997.
[38] VLSI Technology, inc., "Datasheet book: 0.6-micron 5-V core mixed 5 V/3 V I/O cell-based libraries," VLSI Technology, Inc., San Jose, CA, Aug. 1994.
[39] M. van Swaaij, F. Franssen, F. Catthoor, and H. De Man, "Automating high-level control flow transformations for DSP memory management," in *Proc. IEEE Workshop on VLSI Signal Processing*, Napa Valley, CA, Oct. 1992.



Erik Brockmeyer received the degree in electrical engineering in 1998 from the University of Eindhoven, The Netherlands. In the last year he did his thesis on MPEG-4 in the System Exploration for Memory and Power (SEMP) group, University of Eindhoven.

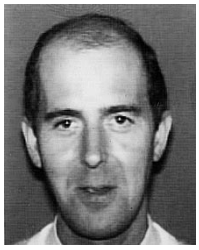
He is now with the DESICS division of the Interuniversity Microelectronics Center (IMEC), Leuven, Belgium. Currently, he is working in this group, headed by F. Catthoor, in the field of data transfer and storage exploration (DTSE), with emphasis on storage cycle budget distribution.



Lode Nachtergaele received the degree of Industrial Engineer in 1989 from the Katholieke Hogeschool Oostende, Oostende, Belgium.

In the same year, he joined the Interuniversity Microelectronics Center (IMEC), Leuven, Belgium, starting his career in the group working on the Cathedral-II silicon compiler. He was involved in the development of the Silage simulator S2C. In 1992, he joined the System Exploration for Memory and Power (SEMP) group. Together with his colleagues, he worked on the ATOMIUM methodology, partially supported by prototype tools. In 1996, he joined the Multimedia Image Compression Systems (MICS), Design Technology for Integrated Information and Communication Systems (DESICS) division, IMEC. In 1999, he became the Operating Officer of the MICS group.

Mr. Nachtergaele is the Belgium head of delegation for the ISO/IEC JPEG standardization committee.



Francky V. M. Catthoor (M'87) received the engineering degree and the Ph.D. in electrical engineering from the Katholieke Universiteit Leuven, Leuven, Belgium, in 1982 and 1987, respectively.

From September 1983 until June 1987, he was a Researcher in the area of VLSI design methodologies for Digital Signal Processing, with Prof. H. De Man and Prof. J. Vandewalle as Ph.D. thesis advisors. Since 1987, he has headed several research domains in the area of high-level and system synthesis techniques and architectural methodologies, all within the Design Technology for Integrated Information and Telecom Systems (DESICS—formerly VSDM) division, Interuniversity Microelectronics Center (IMEC), Leuven, Belgium. He has been Assistant Professor, Electrical Engineering Department, Katholieke Universiteit Leuven, since 1989. His current research activities belong to the field of architecture design methods and system-level exploration for power and area, mainly oriented toward memory management and global data transfer optimization. The major target application domains are real-time signal and data processing algorithms in image, video and end-user telecom applications, and data-structure-dominated modules in telecom networks. Both customized architectures and programmable (parallel) multimedia processors are targeted.

Dr. Catthoor he received the Young Scientist Award from the Marconi International Fellowship Council in 1986. Since 1995, he has been an Associate Editor for the IEEE TRANSACTIONS ON VLSI SYSTEMS, and since 1996, for the *Journal of VLSI Signal Processing*. In 1997, he became a member of the steering board for the VLSI Technical Committee of the IEEE Circuits & Systems Society. He was the program chair of the 1997 International Symposium on System Synthesis (ISSS) and is the general chair for the 1998 ISSS.

Jan Bormans (M'97) received the electrical engineering degree and the Ph.D. degree in applied sciences from the Vrije Universiteit Brussel (VUB), Belgium in 1992 and 1998, respectively.

In 1992 and 1993, he has been a Researcher on image compression at the ETRO laboratory of the VUB. In 1994, he joined the VLSI System and Design Methodologies (VSDM) division, Interuniversity Microelectronics Center (IMEC), Leuven, Belgium. Since 1996, he has been heading the Multimedia Image Compression Systems group in DESICS (Design Technology for Integrated Information and Communication Systems), focusing on the efficient design and implementation of systems-on-a-chip for advanced multimedia applications.

Dr. Borman is the Belgium head of delegation for the ISO/IEC MPEG standardization committee.

Hugo J. De Man (F'86) was born in Boom, Belgium, on September 19, 1940. He received the electrical engineering degree and the Ph.D. degree in applied sciences from the Katholieke Universiteit Leuven, Leuven, Belgium, in 1964 and 1968, respectively.

From 1969 to 1971, he was at the Electronic Research Laboratory, University of California, Berkeley, as an ESRO-NASA Postdoctoral Research Fellow, working on computer-aided device and circuit design. In 1971, he returned to the University of Leuven as a Research Associate of the NFWO (Belgian National Science Foundation). In 1974, he became a Professor at the University of Leuven. During the winter quarter of 1974–1975, he was a Visiting Associate Professor, University of California, Berkeley. From 1984 to 1995, he was Vice-President of the VLSI systems design group of the Interuniversity Microelectronics Center (IMEC), Leuven, Belgium. Since 1995, he has been a Senior Research Fellow of IMEC, responsible for research in system design technologies.

Prof. De Man is a corresponding member of the Royal Academy of Sciences, Belgium, and a member of the Royal Flemish Engineering Society (KVIV).