Systematic Speed-Power Memory Data-Layout Exploration for Cache Controlled Embedded Multimedia Applications

M. Miranda C. Ghez C. Kulkarni F. Catthoor* D. Verkest

IMEC Lab., Kapeldreef 75, Leuven, Belgium
*Also Professor at the Katholieke Universiteit Leuven, Belgium

{miranda,ghez,kulkarni,catthoor,verkest}@imec.be

ABSTRACT

The ever increasing gap between processor and memory speeds has motivated the design of embedded systems with deeper cache hierarchies. To avoid excessive miss rates, instead of using bigger cache memories and more complex cache controllers, program transformations have been proposed to reduce the amount of capacity and conflict misses. This is achieved however by complicating the memory index arithmetic code which results in performance degradation when executing the code on programmable processors with limited address capabilities. However, when these are complemented by high-level address code transformations, the overhead introduced can be largely eliminated at compile time. In this paper, the clear benefits of the combined approach is illustrated on two real-life applications of industrial relevance, using popular programmable processor architectures and showing important gains in energy (a factor 2 less) with a relatively small penalty in execution time (8-25%) instead of factors overhead without the address optimisation stage. The results of this paper leads to a systematic Pareto optimal trade-off (supported by tools) between memory power and CPU cycles which has up to now not been feasible for the targeted systems.

Categories and Subject Descriptors

B.3.2 [**Hardware**]: Memory structures—*Cache memories*; D.3.4 [**Software**]: Programming languages—*Optimisation*

Keywords

Address generation, direct mapped cache, embedded systems, multimedia applications

1. INTRODUCTION

Multi-media systems such as medical image processing and video compression algorithms, typically use a very large amount of data storage and transfers. This is especially a problem for low-power embedded system realisations because the memories and bus transfers are responsible for most of system energy waste (between 50 and 80%). Therefore, optimising the global memory accesses of an application in a so-called Data Transfer and Storage Exploration (DTSE) [1] stages is a crucial task for achieving low power realisations. These optimisations have both platform independent and platform specific phases. The first phase has a positive influence on both energy and performance (irrespective of the architecture) since it removes redundant data transfer and storage and it improves the locality of production/consumption of the data. However, this phase need to be complemented by a platform specific one to exploit the opportunities created. This is especially true for highly constrained architectures like those found in programmable platforms due to resource limitations.

In multimedia, the vast majority of programmable processor use cache controllers to manage their on-chip memories. These controllers decide at run-time when and how the data is copied from the higher memory hierarchy layers to the cache architecture [2]. In this way the designer does not have to worry about managing the memory hierarchy, especially for the dynamic part of the application. However, this can become a run-time bottleneck especially due the cache misses when the required data is no longer present in the on-chip memory. These misses translates into an excess of system bus load and energy because of the traffic of data between main memory and the cache (hierarchies). This results into cycles wasted when the processor is stalled waiting for the data to arrive and in energy spent in reloading the cache. Moreover, it limits the performance of shared-memory multiprocessor based implementations.

All this unnecessary overhead can be avoided at the program level. However, it requires knowledge of the underlying platform architecture. This knowledge can be exploited using platform-aware transformations. Optimisations oriented to minimise the miss rate in the data cache (D-cache) are those exploiting the limited lifetime of the data during program execution (namely inplace optimisations which can reduce capacity misses [3, 4]) and those exploiting the memory data organisation freedom (to remove the conflict misses, e.g. [5, 6]). This reduction in miss rate leads to important savings in energy (see Section 3). However, they typically require to re-write the index expressions of the affected data arrays, thus resulting in a much more complex addressing functionality. This potentially large overhead in addressing can lead to factors overhead in execution time on programmable processors. In this way it can limit a more efficient cache utilisation.

After cache oriented code transformations, the new addressing code is dominated by index expressions of piece-wise linear nature and it becomes a bottleneck for the auto-increment nature of most commercial Address Calculation Unit (ACU) architectures. Therefore, it should be removed before going to the more pointer level

ISSS'01, October 1-3, 2001, Montréal, Québec, Canada

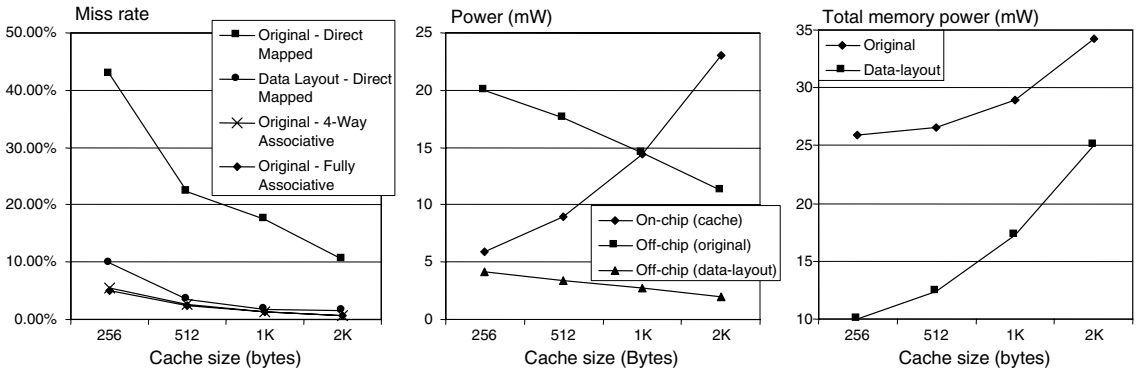

Figure 1: D-cache miss-rate evolution for the MPEG4 Motion Estimation driver for different cache sizes and controller types, including off-chip/on-chip power breakdown and total power evolution for different cache sizes (for a direct mapped cache controller).

specific address optimisation techniques like those found in conventional [7] or state-of-the-art compilers [8]. Moreover, conventional automated address optimisation approaches [9, 10, 11, 12, 13] do not allow this for address code containing modulo/division operations.

To remove this large bottleneck we have developed efficient address optimisation techniques. These are based on the use of program transformations which are largely independent of the targeted instruction-set architecture [14, 15], but which need also to be complemented by more architecture specific ones [16]. Using these techniques, the addressing code is globally optimised, resulting in factors overall improvement in execution cycles when compared to their original data-organised versions.

The clear benefits of the combined approach to aggressively improve cache utilisation by source code transformations which introduce complex addressing which is then reduced again by a high-level address optimisation stage, has to our knowledge not been studied in the literature. We will provide a systematic approach to achieve that in this paper. It leads to Pareto optimal speed power trade-off curves. That approach is illustrated on two real-life application drivers of industrial relevance, using three popular programmable processor architectures, showing important gains in cycle count and energy consumption.

2. EXPERIMENTAL SETUP

Due to flexibility requirements of the implementation an instruction-set processor architecture is preferred over a fully custom one. The architectures which we target in this paper, consist of multi-media (extended) processors (such as TriMedia's TM1000, Intel's Pentium-III MMX) but also more general RISC architectures such as Hewlett-Packard's PA800 architecture.

For realistic embedded implementation we assume their "core" versions which have a relatively small local (L1) on-chip cache memory to reduce the energy per L1 cache access (which is our focus here). These processors are connected to a (distributed) shared off-chip memory (hierarchy). Therefore, meeting real-time constraints in such architectures, requires an optimal mapping of the application onto the target platform.

For our experimental setup we have used the SimpleScalar simulator tool set [20] for simulating cache performance for varying cache sizes. The cache is simulated using the faster and functionally correct sim-cache simulator for measuring cache performance. Existing processors have been used to measure the performance values. The code versions have been compiled and run on the different platforms using the native compilers and enabling their most aggressive speed oriented optimisation features.

Our exploration approach is illustrated on two real-life application drivers: a Cavity Detector algorithm and MPEG4 Full Search Motion Estimation kernel for CIF video frames. The Cavity Detector is an image processing application used mainly in the medical field for detecting tumour cavities in computer tomography pictures (as large as 1280×1000 pixels) [17].

When these algorithms are mapped onto an embedded system, the system bus load and power requirements are quite high. At our laboratory, much effort has been already spent on optimising both drivers at the source-level code, mainly by applying the platform independent transformation stage of the DTSE methodology [18, 19]. In this paper we will focus only on the data cache related speed-power trade-offs during the platform specific mapping stage.

3. MEMORY DATA-LAYOUT OPTIMISATION FOR D-CACHES

A *cache miss* happens whenever a block of data requested by the CPU is not found in the cache. Therefore, the fist time a block has be copied into the cache can also be considered as a miss (namely a *compulsory miss*). However, only those misses related to blocks that have been previously allocated in the cache lead to an overhead. In this case, two possible situations exist [2]: (1) when the cache cannot contain all the blocks needed during execution, *capacity misses* will occur because of blocks being discarded and later retrieved; (2) if the block placement strategy is set associative or direct mapped, *conflict misses* (in addition to compulsory and capacity) will occur because a block can be discarded and later retrieved if too many blocks map to its set.

For the D-cache, optimisations are available which are oriented to minimise both capacity and conflict misses. Capacity misses can be greatly reduced by exploiting the limited life-time of the data (inplace oriented optimisations [3]) and the conflict misses by applying data-layout oriented transformations [6]. Using prototype tools supporting these techniques [21], we obtain the transformed C code. Thus, we now have the initial (reference) and the data layout transformed codes.

After cache simulations we have observed that both drivers are dominated by misses in the D-cache (the miss rate for I-cache is about three orders of magnitude smaller than for D-cache). Therefore, we have focused on optimising the D-cache miss rate by applying main memory data-layout oriented program code transfor-

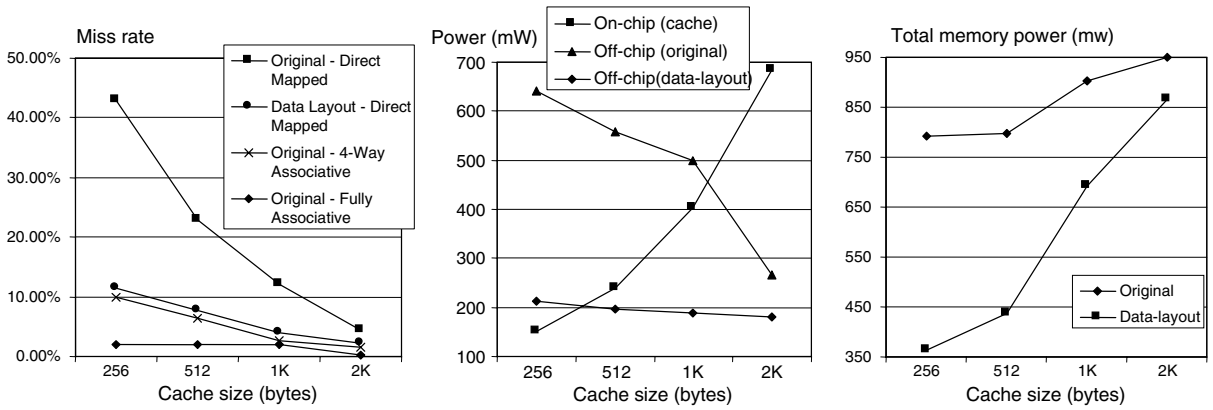

Figure 2: Data-cache miss-rate evolution for the Cavity Detector driver for different cache sizes and controller types, including off-chip/on-chip power breakdown and total power evolution for different cache sizes (for a direct mapped cache controller).

mations. Simulations have been performed using different cache controller types: a 4-way associative cache for the non optimised version and a direct mapped for both optimised and non-optimised versions. Also, different capacity sizes (with the number of cache lines ranging from 256 till 2024) have been evaluated. Figure 1 and 2 shows the miss rate after simulations for both drivers and controller types.

For the non-optimised code, the miss rate obtained using a direct mapped controller is rather large when we compare it to the one obtained when using 4-way associative controller. However, when using the cache optimised code, the miss rate behaves equally well for both controller types. This is clearly an important result because it enables the use of the simpler direct mapped cache controllers for embedded applications, instead of the more costly, and power hungry, set-associative controller types.

4. HIGH-LEVEL OPTIMISATION OF THE INDEX/ADDRESS COMPUTATION

Cache oriented transformations require the introduction of complex modulo and integer division operations in the index expressions. This is because, transformations exploiting the limited lifetime of the data requires complex modulo operations in the address code to express in time the folded nature of the relation between the original and the transformed address spaces in main memory.

On the other hand, data-layout oriented transformations require to create gaps of unused memory locations within array of data allocated in main memory, as well as to allocate in memory different arrays in an interleaved manner. Therefore, these data-layout oriented transformations also require modifications in the original addressing code by introducing complex modulo and integer division operations. Figure 3 illustrates the effect of such transformations on a piece of code representative of the Cavity Detector driver. The combined outcome is that even nested modulo expressions are present.

As a result, for both application drivers described in Section 3, a large overhead in addressing is created. For both drivers, this overhead is mostly due to the more complex index expressions (mainly of piece-wise linear nature) introduced during the inplace [3] and data-layout [6] DTSE oriented optimisation steps. Indeed, a considerable number of costly modulo operations are executed and traditional compilers completely fail to efficiently eliminate them. Hence, the execution time increases considerably: a factor between 2-20 depending on the application and the target processor architecture (see Section 5).

```
for(x=1; x<N-2; x++) {
  for(y=1; y<M-2; y++) {
    for(k=-1; k<1; k++) {
      ....
      A[x][y] += B[x+k][y]*C[abs(k)];
      ....
    }
    A[x][y] /= tot; }}
                    (a)

for(y=0;y<M+2;y++) {
  for(x=0;x<n+2;x++) {
    ....
    if(x>=0 && x<N && y>0 && y<M-1)
      D[x%3] = B[(y*N+x%3)%160+(y*N+x%3)/160*256+ 96];
    ....
    if(x-1>=1 && x-1<=N-2 && y>=1 && y<=M-2)
      for(k=-1;k<=1;k++)
        acc+=D[(x-1+k)%3]*C[abs(k)];
    acc/=tot; }}
                    (b)
```

Figure 3: Illustration of data cache oriented transformations in the Cavity Detector driver: (a) initial code; (b) transformed code.

Our novel design script allows to remove this overhead at the source-level code by reducing the complexity and the amount of address operations [16]. This high-level optimisation stage consists on two main stages.

The first one exploits algebraic and modulo properties to reduce the number operation instances. This stage does a re-factorisation of addressing expressions [22] (containing also modulo/division operations) that compilers are not capable to perform (see Fig 5b). This factorisation typically exposes more opportunities for common sub-expression elimination than those originally present in the initial code. When complementing this with aggressive code hoisting techniques across both loops and conditionals, the number of executed operations is minimised.

At that point, the second stage can focus on reducing the execution cost of the (remaining) modulo/division operations by transforming these onto a efficient combination of linear induction variables and conditional code (see Fig 5c) [16].

High-level address transformations also have a positive effect on

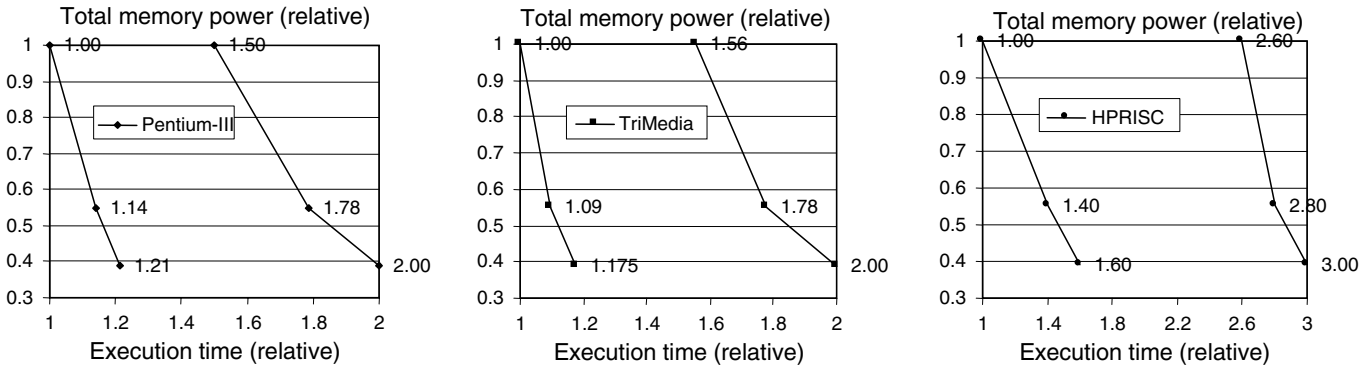

Figure 4: Power and performance trade-offs for the Motion Estimation kernel for different processors and memory data-layouts. Lines at the right represent trade-offs before the address optimisation phase and at the left after it.

memory accesses and in L1-misses both for I-caches and D-caches. For the D-cache, we have observed for both drivers (see Section 2) that the amount of data memory accesses as well the number of data misses decreases slightly ($\simeq$<10%). However, this improvement in data memory power is very limited when compared by the one achieved using the memory data-layout oriented transformations (see Section 5). For the I-cache, a larger reduction in instruction memory fetches (>40%) is obtained (due to the more efficient addressing code) while the reduction in instruction fetch misses stays also limited ($\simeq$<10%). Still, in both drivers the miss-rate for the I-cache is about three orders of magnitude smaller than for the D-cache and so the off-chip memory related power.

```
for(y=0;y<10;y++){
  for(x=0;x<100;x++){
    if(x>1) A[(y%3)*3
              +(x-2)%3]=...
    if(x>4) ...=A[(y%3)*3
              + (x-5)%3]
}}
```

(a) Initial

```
for(y=0;y<10;y++){
  v_y = (y%3)*3;
  for(x=0;x<100;x++){
    v_yx = (x-2)%3 + v_y;
    if(x>1) A[v_yx] = ...
    if(x>4) ... = A[v_yx];
  }}
```

(b) Index expression optimisation

```
for(y=0;y<10;y++){
  if(p_y>=9) p_y-=9;
  for(x=0;x<100;x++){
    if(p_x>=3) p_x-=3;
    v_yx = p_x + p_y;
    if(x>1) A[v_yx] =...
    if(x>4) ...= A[v_yx];
    p_x++;}
  p_y+=3;}
```

(c) Modulo operation sustitution

Figure 5: Illustration of processor independent address transformations for the Cavity Detector driver: (a) initial code; (b) after algebraic and modulo oriented transformations; (c) after modulo substitution transformations.

5. TRADE-OFF BETWEEN PERFORMANCE, POWER AND MAIN-MEMORY SIZE

We now discuss how the above technique can be applied to achieve important system level trade-off's. For both drivers we have started with an initial algorithm which has been optimised from a DTSE perspective.

In the first step, we obtain miss rates for different cache sizes when using a cost efficient direct mapped cache. This is shown in Figure 1 for the Motion Estimation and in Figure 2 for the Cavity Detector. Once we have obtained the miss rates for different cache sizes, we compute the total (off-chip plus on-chip) required power for each driver. We observe that for both drivers a cache size with 256 bytes consumes the least total power. For smaller cache sizes, the total consumed power would be dominated by the accesses (due to misses) to the off-chip memory. For larger sizes the accesses to the on-chip memory dominates the total power though.

In a next step, we choose the power optimal cache size of 256 bytes to decide on a trade-off between main-memory size (the overhead in memory space in the data layout optimisation process as compared to the initial algorithm) and the reduction in miss rate as shown in Tables 1 and 2. Thus depending on the design constraints, the designer can now either choose a lower power solution with some overhead in main-memory size and vice-versa. For the Cavity Detector algorithm we have decided to choose two versions representative of extreme points in the search space and observe the trade-offs: one with a 2% main-memory size overhead (labelled "Light") and one with a factor of 2 overhead (labelled "Aggressive"). For the Motion Estimation kernel the points selected are responsible for a 30% overhead in main-memory size (labelled "Light") and a much larger factor 3 overhead (labelled "Aggressive").

In the final stage, we perform address optimisations (also supported by prototype tools [16]) to remove the overhead in addressing operations introduced in the data layout optimisation process. Figures 4 and 6 show that we are able to largely remove this overhead in addressing. This is visible by comparing the projected values on the horizontal axis of the curve before the high-level address optimisation phase (right-hand side) and after it (left-hand side).

For both drivers, using their "Light" data-layout version, we are able to reduce memory related power by almost a factor 2. For the Motion Estimation, this is achieved by trading-off a 30% in main-memory size with a 14-40% in execution time (depending on the target processor), instead of the 80-280% CPU cycle overhead

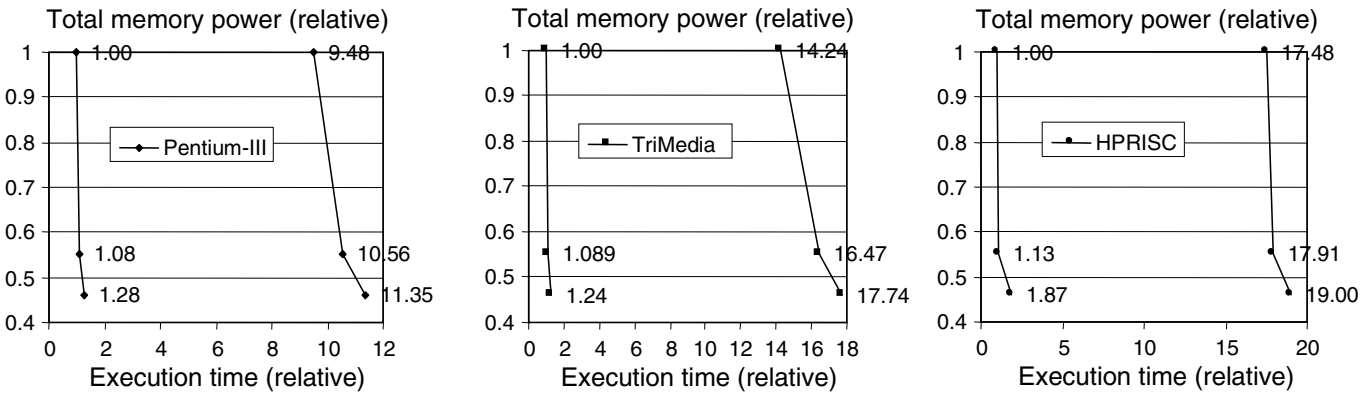

Figure 6: Power and performance trade-offs for the Cavity Detector algorithm for different processors and memory data-layouts. Lines at the right represent trade-offs before the address optimisation phase and at the left after it.

without the address post-processing stage (see Figure 4). For the Cavity Detector algorithm, main-memory size overhead is limited to just a 2% and the overhead in execution time is less 13% (irrespective of the target processor) instead of the factor 10-20 overhead in executed cycles when not using the address post-processing stage (see Figure 6). We assume that the data related memory power does not significantly change with the address code transformations stage but mainly with the memory data-layout stage as motivated in Section 4.

Table 1: main-memory size/power/speed trade-offs for the Motion Estimation.

Data-layout trade-off	Memory-size Overhead	Avrg.Power Gain	Avrg.Speed Degradation
Light	30 %	45 %	14-40 %
Aggressive	3X	60 %	18-60 %

Table 2: main-memory size/power/speed trade-offs for the Cavity Detector.

Data-layout trade-off	Memory-size Overhead	Avrg.Power Gain	Avrg.Speed Degradation
Light	2 %	45 %	8-13 %
Aggressive	2X	65 %	24-87 %

Note that the approach we have used in this work is mostly power centric namely, we first optimise for power (by selection the optimal cache size) then for main-memory size (by deciding the amount of data-layout) and lastly for performance (by optimising the addressing code). But the above technique can be used for a performance centric approach too. In this case, the designer should first choose the optimal cache size for data miss rate (e.g. by selecting the cache size from which the decrease in number of misses starts to saturate) and then select the amount of data-layout corresponding to the required trade-off between main-memory size and memory power. Following this approach, cache sizes of 512 bytes for the Motion Estimation and 1024 bytes for Cavity Detector would lead to better system performance (less data misses) at the expenses of higher data memory power (see total memory power evolution in Figures 1 and 2). Still, Pareto curves like those shown in Figure 4 and 6 can be used in both scenarios.

6. CONCLUSIONS

In this paper we have shown how to efficiently combine memory data-layout and address program transformations in a systematic way to trade-off speed and power when mapping data-intensive applications into architectures containing cache memories. The clear benefits of the combined approach have been illustrated by two real-life applications of industrial relevance, using popular programmable processor architectures. Important gains in energy (up to a factor 2 less) with very limited impact on execution time ($<$25%) are reported instead of the factors overhead in execution time without the address post-processing stage. Alternatively, execution time gains be achieved at the system level with a less aggressive power gain while trading-off main-memory size with total memory power.

7. REFERENCES

[1] F.Catthoor, K.Danckaert, C.Kulkarni, T.Omnes, *Data transfer and storage architecture issues and exploration in multimedia processors*, book chapter in "Programmable Digital Signal Processors: Architecture, Programming, and Applications" (ed. Y.H.Yu), Marcel Dekker, Inc., New York, 2000.

[2] D.Patterson, J.Hennessey, "Computer architecture : A quantitative approach", *Morgan Kaufmann Publ.*, San Francisco, 1996.

[3] E. De Greef, F. Catthoor, H. De Man, *Program Transformation Strategies for Memory Size and Power Reduction of Pseudo-regular Multimedia Subsystems* IEEE Trans. on Circuits and Systems for Video Technology Vol. 8, No. 6, pp. 719-733, October 1998.

[4] C.Kulkarni, F.Catthoor, H.De Man, "Hardware cache optimization for parallel multimedia applications", *Proc. EuroPar Conference*, Southampton, U.K.,pp.923-931, Sep. 1998.

[5] P.Panda, F.Catthoor, N.Dutt, K.Danckaert, E.Brockmeyer, C.Kulkarni, A.Vandecappelle, P.G.Kjeldsberg, "Data and Memory Optimizations for Embedded Systems", *ACM Trans. on Design Automation for Embedded Systems (TODAES)*, Vol.6, No.2, pp.142-206, April 2001.

[6] C.Kulkarni, F.Catthoor, H.De Man, Advanced Data Layout Optimization for Multimedia Applications, Proc. Workshop on Parallel and Distributed Computing in Image, Video and Multimedia Processing (PDIVM'00), IPDPS'2000, May 2000.

[7] A.V.Aho, R.Sethi, J.D.Ulmann, “Compilers: principles, techniques and tools”, Addison-Wesley, 1986.

[8] S.Muchnick, *Advanced Compiler Design and Implementation* Morgan Kaufmann Publishers, 1997.

[9] R.Leupers, P.Marwedel, *Algorithms for Address Assignment in DSP Code Generation*. Proc. Intnl. Conf. on Computer-Aided Design (ICCAD), 1996.

[10] C.Liem, P.Pauling, A.Jerraya, *Address calculation for retargetable compilation and exploration for instruction-set architectures*. Proc. Design Automation Conference (DAC), pp. 597-600, 1996.

[11] C.Gebotys, *DSP address optimisation using a minimum cost circulation technique*. Proc. Intl. Conf. on Computer-Aided Design (ICCAD), pp. 100-104, 1997.

[12] A.Sudarsanam, S.Liao, S.Devadas, *Analysis and evaluation of address arithmetic capabilities in custom DSP architectures*. Proc. Design Automation Conference (DAC), 1997.

[13] S.Udayanarayanan, C.Chakrabarti, “Address Code Generation for Digital Signal Processors”, *Proc. ACM Design Automation Conf.*, pp. 353-358, Las Vegas, June 2001.

[14] M.Miranda, F.Catthoor, M.Janssen, H.De Man, “High-level Address Optimisation and Synthesis Techniques for Data-Transfer Intensive Applications”, *IEEE Trans. on VLSI Systems*, Vol.6, No.4, pp.677-686, Dec. 1998.

[15] S.Gupta, M.Miranda, F.Catthoor, R.Gupta, *Analysis of High-level Address Code Transformations for Programmable Processors*, In Proc. Design and Test in Europe Conf. (DATE), 2000.

[16] C.Ghez, M.Miranda, A.Vandecappelle, F.Catthoor and D.Verkest, Systematic high-level Address Code Transformations for Piece-wise Linear Indexing: Illustration on a Medical Imaging Algorithm, *In Proc. Workshop on Signal Processing Systems (SIPS)*, 2000.

[17] M.Bister, Y.Taeymans, J.Cornelis, “Automatic Segmentation of Cardiac MR Images”, *Computers in Cardiology*, IEEE Computer Society Press, pp.215-218, 1989.

[18] K. Danckaert, F. Catthoor, H. De Man, *Platform independent data transfer and storage exploration illustrated on a parallel cavity detection algorithm*. Proc. PDPTA (CSREA Conf. on Parallel and Distributed Processing Techniques and Applications), 1999.

[19] Kristof Denolf, Peter Vos, Jan Bormans and Ivo Bolsens, Cost-efficient C-Level Design of an MPEG-4 Video Decoder *Proc. Workshop on Power and Timing Modeling, Optimization and Simulation*, 2000.

[20] D.Burger, T.Austin, “The Simplescalar Toolset”, version 2.0, online document available at http://ww.cs.wisc.edu/ mscalar/simplescalar.html.

[21] C.Kulkarni, M.Miranda, C.Ghez, F.Catthoor, H.De Man, “Cache Conscious Data Layout Organization For Embedded Multimedia Applications”, *Proc. 4th ACM/IEEE Design and Test in Europe Conf.*, Munich, Germany, March 2001.

[22] J.M.Janssen, F.Catthoor, H.De Man, *A specification invariant technique for operation cost minimisation in flow-graphs*, Proc. Intln. Symp. on High-level Synthesis, pp. 146-157, 1994.