LOW-POWER IMPLEMENTATION OF AN OFDM BASED CHANNEL RECEIVER IN REAL-TIME USING A LOW-END MEDIA PROCESSOR

P. Op de Beeck, C. Ghez, E. Brockmeyer, M. Miranda, F. Catthoor and G. Deconinck

IMEC. Kapeldreef 75, 3001 Leuven, Belgium

ABSTRACT

The implementation of advanced channel receivers using low-end multimedia instruction set processors is a productive, flexible and cost effective alternative to custom hardware. The stringent real-time and low-power requirements become attainable on condition that for these applications the impact of the data transfer and storage related issues is first drastically reduced.

This paper illustrates the implementation of a real-time and low-power OFDM based channel receiver on a TriMedia TM1300 processor. This is the result after applying our Data Transfer and Storage Exploration methodology. In particular we focus on the exploration of data formating alternatives of background memory for efficient sub-word level acceleration. The outcome of our approach is an optimized source code description of the channel receiver which optimally exploits the existing memory hierarchy while making use of the available SIMD instructions in a cost effective manner. Following this approach we have achieved more than an order of magnitude reduction in energy consumed in the memory hierarchy while executing in real-time. Moreover, the slack achieved in execution time makes it possible to lower the frequency, and thus the V_{dd} of the CPU core towards the minimum recommended by the processor's specification. This allows an extra 36% energy reduction in the CPU core.

1. INTRODUCTION

In the near future mobile radios will be equipped with Digital Audio Broadcasting (DAB) reception. A DAB broadcaster is able to provide a combination of services up to a total of about 1.8Mbps [1, 2]. The transmission system in the DAB standard is based on an Orthogonal Frequency Division Multiplex (OFDM) transportation scheme using up to 1536 carriers (Mode I) for terrestrial broadcasting. Other standards making use of OFDM include ADSL, 802.11a and DTV.

The implementation of such advanced channel receivers using low-end multimedia instruction set processors is a productive, flexible and cost effective alternative to custom hardware. We have chosen the TriMedia TM1300 [6] because of its sub-word level or SIMD processing capabilities. This combined with the VLIW nature of its instruction set architecture gives large room for exploiting concurrency at the instruction level, hence for further speed-up.

Unfortunately, the exploration of the application specific SIMD acceleration capabilities is not supported by Trimedia's compiler and it typically requires the designer to manually insert calls to optimized assembly libraries at the source code level. Still, even if this is done so, the data format required for the SIMD operation does not typically match that one chosen for background data storage. The consequence is that the potentially obtainable speed-up is hidden by the cycle overhead of the extra data formating instructions (e.g. SIMD (un)pack operations). These extra formatting operations are necessary to combine/split the data words before/after these are processed by the SIMD unit. Our goal is to show how that overhead can be avoided by adapting the data storage organization in a platform dependent way. This is complemented with background memory data-layout techniques for improved cache hit behavior and some high level address- and processor specific optimizations.

In this paper we show how we can obtain a real-time and low-power implementation of a DAB channel receiver on the TriMedia TM1300 processor. We firstly give an overview of how the data transfer and storage related implementation bottlenecks, which are truly target implementation independent, have been eliminated. The main focus however is on the platform dependent exploration of background data format alternatives for the OFDM block. These formatting transformations allow to reduce the number of cache misses and to efficiently utilize the special instruction set provided in the TM1300.

At the DAB decoder side the OFDM carrier spectrum is reconstructed by doing a forward 2048-point FFT (Mode I) on the received OFDM symbol. In our reference code, an optimized industrial C code implementation of the standard, this kernel takes up 57% of the total execution time, measured by compiling and running this reference code using TriMedia's native tool set. In total this has resulted in 3.7 seconds execution time for a 1 second input stream, thus

not achieving real-time. Likewise, this kernel is responsible for the dissipation of 60% of the total data memory energy consumption of $500mJ$.

Following the illustrated approach we have achieved more than an order of magnitude reduction in energy consumed in the data memory hierarchy while achieving real-time execution. Moreover, the slack obtained in execution time makes it possible to lower the frequency, and thus the V_{dd} of the CPU core towards the minimum recommended by the processor's specification. This allows an extra 36% energy reduction in the CPU core. We have estimated that an 82% energy reduction has been achieved in the platform independent optimized version of the application code. This is reduced further to become only 6.25% in the final version after the exploration of platform dependent alternatives or in other words a factor 16 gain in data memory energy consumption when compared to the initial reference.

2. RELATED WORK

In 1995 the first instruction set processors have been introduced with support for SIMD operations [17, 18]. These operations apparently strengthen the interaction between the data path and the way data is stored in memory. From a methodological standpoint this is unfortunate because the background memory problem should better be decoupled from data path issues.

In [14] the data packing is tackled during the code selection phase. Only the data needed together is packed together in background memory. However, no prior code transformations are explored to create more freedom. Instead, a constraint integer linear problem is formulated which enforces a particular horizontal data layout. Recently [15] has proposed a method that horizontally orders the scalar data during address assignment to improve program performance in SIMD processors. The use of a variable access graph obtained after operation scheduling is required. This however limits the freedom during memory management in general and for background data merging in particular [13].

Our approach reverses this bottom-up strategy and as we show in this paper there is still enough freedom for efficient SIMD level operation scheduling. Interesting in this context is the work in [16] where it is shown that the interaction between SIMD exploitation and (background) memory transformations can be decoupled.

Transformations for merging signals within background memory have been addressed in different contexts. For instance in [10] memory accesses are coalesced in order to more efficiently use a processors memory system. The proposed algorithm is limited to merging consecutive memory references, which are being exposed by loop unrolling. Furthermore, runtime checks are inserted to determine alias and misalignment hazards. In our approach these tests are obsolete because data merging happens before data is assigned to any specific memory address and pointers are avoided. In [11] scalar data is clustered into larger words to minimize the number of compulsory misses in the data cache. Both previous techniques are limited to scalar level merging. Moreover, they require the scheduling to happen beforehand with the limitations mentioned above. Another application for background data merging is described in [12]. The goal is to exploit wide busses with explicit wide load and store operations. The algorithm works with an extended dependence graph and is applied before modulo scheduling.

Finally, also in the custom processor domain data merging has been explored [13]. The main goal is to reduce the number of memory accesses and additionally it also reduces the bit waste. This approach is targeted toward dynamically allocated record types. The additional freedom of manifestly specified arrays is not looked at. Both approaches do not consider constraints coming from the (SIMD) operation format which is the focus of this paper.

3. PLATFORM INDEPENDENT OPTIMIZATIONS

In this section we give an overview of the platform independent Data Transfer and Storage Exploration (DTSE) carried out on the full DAB decoder. These stages already give a large cost reduction compared to a non-trivial industrial ASIC implementation [19] implementing such functionality. A simplified view off the DAB decoder before and after the platform independent stages is shown in Fig. 1.

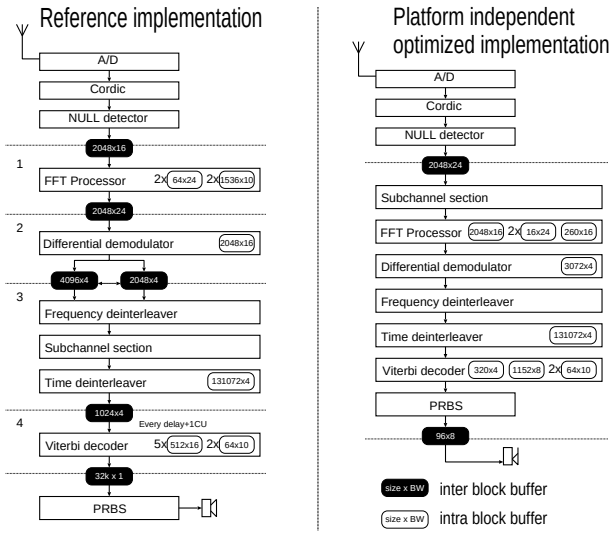

Figure 1. The DAB channel receiver before and after the platform independent DTSE.

The reference model is used as a starting point for the DTSE optimizations. The platform independent phase is

subdivided in three steps. The first step is removing all the redundant accesses by means of data-flow transformations and to break data-flow bottlenecks which limit the freedom for subsequent transformations. One good example in DAB is moving the sub channel selection to the start of the decoding chain, thus only processing OFDM symbols which are really needed.

The next step will improve the global locality and storage size by performing global loop transformations [20]. For instance, in the reference many functional blocks require inner-block communication buffers (see left hand side of Fig. 1) rather then reusing the available buffers inside the blocks (right hand side). This phase is applied aggressively. During the platform dependent phase, where timing is important, buffers will be carefully re-introduced to meet real-time constraints.

Finally a data reuse step will explore the opportunities for making local data copies. There are very few such opportunities in DAB.

The platform independent cost function is defined in terms of access count, (square root of) storage requirements and estimated energy cost [5]. This cost function is appropriate to estimate the energy cost at the high level and to make a relative comparisons between different explorations. The totals that make up this cost function are shown in Table 1.

Implementation	buffer count	size (KBits)	#accesses (10^6)	relative energy
Industrial reference	19	776	1	1
Platform independent	12	636	0.11	0.12

Table 1. Improvements in storage, accesses and energy for platform independent DTSE.

The platform independent DTSE stage has been successfully applied on the DAB channel receiver. According to our high level energy model a relative gain of a factor 8 is obtained compared to an industrial reference. Although the presented stages in this section are not the focus of this paper they must be applied beforehand in order to maximally benefit from the platform dependent optimizations presented next [3].

4. PLATFORM DEPENDENT OPTIMIZATIONS

After compiling and running the platform independent code on the Trimedia TM1300 we observe an 82% gain in data memory energy consumption and about the same in terms of time reduction. However, the FFT kernel now takes up a considerable 72% of the total data memory energy consumption. To reduce this impact the application of platform dependent optimizations have been considered. In concrete, several background data format alternatives for both an efficient utilization of the sub-word level acceleration capabilities as well as an optimal use of the data memory hierarchy of the processor have been explored. Furthermore we have done vertical data layout transformations for cache miss reduction. Finally, high-level address and processor specific optimizations have been carried out. These different steps will be further elaborated on in the following subsections.

4.1. Background data format exploration

OFDM at the receiver side is implemented as a forward complex FFT. The FFT kernel is a butterfly operation requiring 8 data inputs (4 real and 4 imaginary values) and 6 coefficients (3 cosines and 3 sinus values). Figure 2(a) illustrates this.

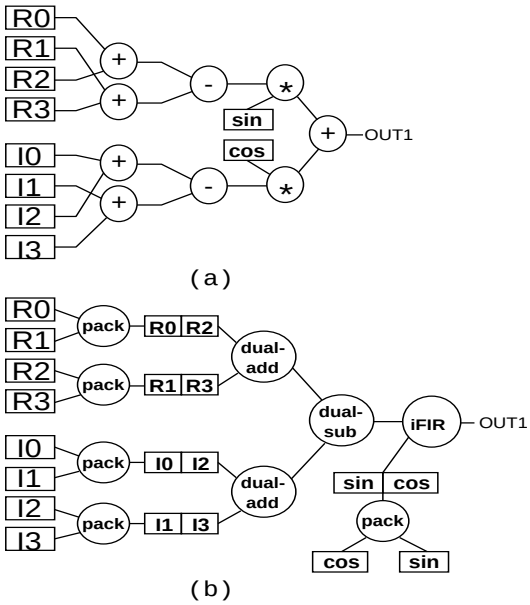

Figure 2. Representative part of the FFT butterfly before (a) and after (b) SIMD acceleration

The regular structure of the FFT butterfly makes it a good candidate for sub-word level acceleration. The mapping onto the Trimedia instruction set is shown in Figure 2(b). The two multiplications and one addition at the end of the butterfly can for instance be performed by one *iFIR16* instruction, outlined in Figure 3. The key feature is to reinterpret the content of a register as two sub-words. In this way the total amount of word-level arithmetic operations has been reduced by more than a factor 2. However, many extra packing (*pack*) instructions are introduced to correctly feed the SIMD operations. This penalty in extra cycles can be avoided when data is already packed in the background memory. With that a large reduction in data accesses is obtained for free. This is not to be confused with byte addressable memory, with sub-word sized content, where the number of cache accesses is not reduced.

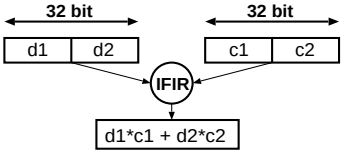

Figure 3. The iFIR16 operation.

Basic group structuring (BGS) is a transformation [5] that horizontally merges the elements of different arrays. Usually these are elements that are processed together. BGS can have a large impact on data memory -power, -size and -bandwidth.

This section describes how the exploration of BGS alternatives has been applied on the FFT kernel. BGS exploration is coupled to the exploitation of SIMD operations since it changes the layout of the arrays, the way they are accessed, and hence it also changes the types of operations associated with them. Since the variety of SIMD instructions in the TriMedia ISA is limited there are only a few BGS alternatives which are interesting to explore. A description of the different Basic Group Structuring opportunities is illustrate in Figure 4.

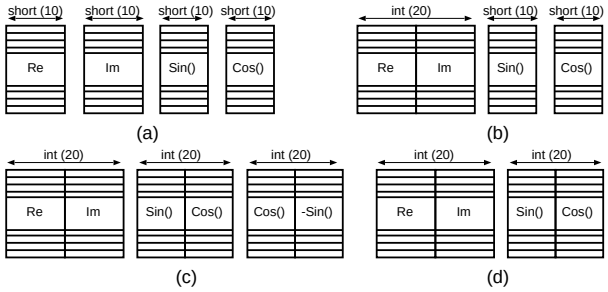

Figure 4. BGS exploration: (a) no BGS at all; (b) partial-1 BGS with packed Re/Im input data; (c) aggressive BGS with Re/Im packing and two (redundant) coefficient arrays; (d) partial-2 BGS with Re/Im packing and one coefficient array. The declared type for each (packed) array is indicated together with the number of useful bits (within brackets).

4.1.1. Partial-1

To compute one FFT butterfly, data from the Real- and Imaginary part are always read and written together. Two 16-bit arrays of 2048 elements are declared in the initial code version, hence two load/store operations are required for transferring these elements to the FUs. After BGS only one load/store remains accessing a 32 bit wide array. The coefficients are untouched in this particular version. However, since we intend to use SIMD operations these values need to packed inside the kernel, incurring an amount of cycles.

4.1.2. Aggressive

To completely eliminate the previously mentioned penalty, we explore an aggressive packing strategy where for all occurring layouts present in the code the coefficient arrays are packed accordingly. In the FFT this results in two packed versions of the coefficient arrays (see Figure 4(c)). One has the format *sin/cos* and the other *-sin/cos*. Clearly all pack- and unpack overhead has been eliminated. On the other hand, one still needs 2 load/store operations to access the coefficients.

4.1.3. Partial-2

In a third version we reduce this number of load/stores to 1 by keeping a single coefficient format (e.g. *sin/cos*). This reintroduces some cycle overhead because to reconstruct the *-sin/cos* format we need to do an unpack, a negate and a repack operation. It is not immediately clear which BGS option is the best. This shows that BGS exploration is strongly related to the type of instructions available in the instruction set. For instance, having a special instruction that negates only one part of a word would be very useful in this particular case.

4.2. Data layout transformations for cache miss reduction

The data cache of the Trimedia TM1300 has a block size of 64 bytes. Hence, each time a cache miss occurs, 16 coefficient values of 4 bytes are transferred from main memory.

In the reference code the stride between two consecutive accesses to the coefficient array is fairly large. This means there is high probability that the 15 other elements, which are brought to the cache anyway, are being flushed before they are actually needed. This is highly undesirable because firstly cycles are wasted because data has to be flushed to make room in the cache and secondly cycles are wasted during the cache miss when the flushed data is later on requested again.

To alleviate this pressure a vertical data layout is applied. This amounts to rewriting address expressions in such a way that the largest multiplicative constant associated to each loop iterator is assigned to the outer loop index and so on. Naturally the array elements need to be reordered accordingly. All BGS versions can benefit from such a transformation, but the problem was especially noticeable in the aggressive case, simply because the cache pressure is higher there.

4.3. High-level address and processor specific optimizations

All code versions have been complemented with high-level address optimizations[8, 9]. These are mainly devoted to

further reduce execution time of cycles spend on address arithmetic computations. In addition some TriMedia specific foreground memory oriented transformations have been considered. More specifically inlining and loop unrolling to move small arrays in the critical path of the FFT kernel to the register file. This last transformation should be applied with care to avoid register spilling.

5. EXPERIMENTAL FRAMEWORK

We have generated different DAB version tracks, each starting from the platform independent code. Each track corresponds to one of the BGS options and contains intermediate versions for all the sub steps in the platform dependent stage. The experiments are carried out by linking the DAB decoder to a test bench that generates a DAB stream of 10 frames (i.e. 1 second). Real-time therefor means an execution time smaller than 1 second.

All versions have been compiled with *-O2* using the native Trimedia compiler and have been executed on the TM1300 development board which has a clock frequency of 166MHz.

Execution time is measured with the standard C library function *clock()*. The cycle and access count measurements are done via hardware counters present on the TM1300 itself. Total energy spend in the data memory is obtained using an energy model defined as

$$E_{tot} = N_{hits} * E_{cache}^{tag+data} + N_{miss} * E_{cache}^{tag} + (N_{miss} + N_{cb}) * E_{sram}$$

with N_{hits}, N_{miss} and N_{cb} the number of cache hits, cache misses and copybacks respectively. $E_{cache}^{tag+data}$ is the energy per access to the cache activating both tag and data lines. E_{sram} is the energy per access to the main memory. We use Cacti [4] with a $.35\mu m$ sram technology to estimate the energy per access.

6. RESULTS

In our experiments we mainly focus on the data memory energy consumption and the overall execution time. We are also interested in the trade-off between the total number of data accesses and the total CPU cycles. These cycles are defined as the total number of execution cycles in the ideal case where there are no instruction or data stalls. It is precisely this trade-off that is being explored during the BGS step. Finally we capture the cache behavior by counting the number of cache misses.

6.1. Exploration of BGS alternatives

We first like to evaluate the effect of BGS applied before SIMD transformations on the FFT data path. For that purpose we obtained 4 versions corresponding to a version where only SIMD transformations have been applied and the other being the three BGS exploration tracks taken right before data layout.

Implementation	Energy (mJ)	Execution time(secs)	# read/writes (10^6)	# misses (10^6)	CPU cycles (10^6)
PI	91.6	0.708	59.67	1.15	87.7
SIMD only	91.7	0.716	59.74	1.15	88.87
Partial-1	53.0	0.559	34.31	1.02	67.97
Aggressive	54.2	0.555	35.12	1.06	67.32
Partial-2	48.2	0.374	20.41	0.71	66.49

Table 2. Impact of BGS on SIMD acceleration, no data layout or processor specific optimizations have been applied

Table 2 shows that all BGS options outperform the SIMD only version, mostly due to a decrease in the number of data accesses. In fact, compared to the platform independent (PI) code, only doing SIMD transformations is slightly worse. This is because of the increased number of CPU cycles coming from extra pack/unpack operations. Because of this poor result we have rejected the SIMD only version from the exploration phase.

The next step is to find out which of the three BGS alternatives is optimal in terms of energy consumed in the data memory and execution time. In Table 3 we list all the BGS tracks after the platform dependent stages. The numbers reported are for the full decoder. The same view is given in Table 4 but for the FFT kernel alone.

Implementation	Energy (mJ)	Execution time(secs)	# read/writes (10^6)	# misses (10^6)	CPU cycles (10^6)
PI	91.6	0.708	59.67	1.15	87.7
Partial-1	32.5	0.383	21.30	0.30	55.98
Aggressive	32.6	0.379	21.38	0.29	55.38
Partial-2	31.1	0.374	20.41	0.25	55.44

Table 3. BGS exploration for the full DAB decoder

Implementation	Energy (mJ)	Execution time(secs)	# read/writes (10^6)	# misses (10^6)	CPU cycles (10^6)
PI	67.1	0.411	43.55	1.07	39.89
Partial-1	6.9	0.075	4.41	0.18	8.33
Aggressive	7.0	0.072	4.50	0.20	7.65
Partial-2	5.5	0.068	3.55	0.15	7.86

Table 4. BGS exploration inside the FFT kernel

We can conclude that the second partial data format results in the best code both in data memory energy and execution time. Even though the total CPU cycles are slightly higher than in the aggressive alternative. In Partial-1 there is even more extra packing going on, in other words even more CPU cycles, but in this case the execution time does become slower. This clearly demonstrates the trade-off between CPU cycles and number of data accesses.

6.2. Impact of platform dependent optimizations

Table 5 and 6 give the breakdown into the different platform dependent sub steps for the Partial-2 alternative.

Implementation	Energy (mJ)	Execution time(secs)	# read/writes (10^6)	# misses (10^6)	CPU cycles (10^6)
Reference	498	3.70	326.5	4.2	599.2
PI	91.6	0.708	59.67	1.15	87.7
+BGS	48.2	0.506	31.37	0.71	66.49
+Data Layout	47.4	0.473	30.92	0.53	66.15
+Proc. Spec.	31.1	0.374	20.41	0.25	55.44

Table 5. Impact of different platform dependent steps for the full DAB decoder

In the first step we introduce BGS on top of SIMD transformations. This reduces the number of data accesses by almost a factor of 2. There is also a 25% gain in CPU cycles due to the smaller number of instructions that have to be executed after SIMD transformations.

During data layout we try to reduce the cache pressure by reordering the accesses to intermediate buffers. The gain is best observed in the aggressive BGS alternative where the 1.05 million cache misses after BGS are reduced to 0.56 million. This number is close to the 0.53 million misses in Partial-2.

The final step has the largest impact because at that stage you can simplify address computations, functions calls can be eliminated by inlining code and small intermediate buffers can be permanently moved to the register file. It should be emphasized that all these optimizations are only made possible by the previous sub steps.

In Table 6 you can verify that the overall gains truly originate from optimizing the FFT kernel. Initially the FFT kernel takes 57% of the overall execution time and 60% of the total energy consumption in the data memory. In the end this share is reduced to 18% in time and about the same in energy. The FFT bottleneck clearly has been removed.

Implementation	Energy (mJ)	Execution time(secs)	# read/writes (10^6)	# misses (10^6)	CPU cycles (10^6)
Reference	299	2.11	195.9	2.58	162.2
PI	67.1	0.411	43.55	1.07	39.89
+BGS	22.6	0.198	14.47	0.60	18.92
+Data Layout	21.7	0.166	14.03	0.42	18.77
+Proc. Spec.	5.5	0.068	3.55	0.15	7.86

Table 6. Impact of different platform dependent steps for the FFT kernel

6.3. Overall gain

Table 7 summarizes the obtained energy reduction in the data memory hierarchy and the final speed-up. In an ASIC design we could achieve a data memory energy consumption of $25mJ$, estimated using a $.35\mu m$ sram power model from an industrial partner. With an energy gain of a factor of 16 on the TM1300 we come close to this number (i.e. $31.1mJ$).

Implementation	Normalized Energy (%)	Normalized Execution time(%)
Reference	100	100
Platform Independent	18.29 (5×)	19.16 (5×)
Platform Dependent	6.25 (16×)	10.12 (10×)

Table 7. Normalized data memory energy and time at the different reference points, within brackets the gain is reported

Since the final version of the channel receiver is only taking 0.37 seconds, the slack achieved in execution time makes it possible to lower the frequency, and thus the V_{dd} of the CPU core towards the minimum recommended by the processor's specification. In the TM1300, V_{dd} can be lowered from $2.5V$ to $2.0V$ [7]. Assuming a linear relationship between frequency and V_{dd} we should already be able to reach the minimum $2.0V$ in the platform independent version. Ultimately this leads to a 36% energy reduction in the CPU core while still running at real-time.

7. CONCLUSION

In this paper, we illustrate the application of a Data Transfer and Storage Exploration for the implementation of a low-power Digital Audio Broadcast channel receiver in real-time using a TriMedia TM1300 processor. We show how the platform independent data transfer and storage related bottlenecks are first eliminated. In a second phase the resulting application source code can be adapted to optimally exploit the available memory hierarchy while efficiently using the sub-word level acceleration capabilities provided by the architecture. These optimizations have brought sufficient speed-up improvements so as to allow real-time execution of the decoder with a factor of 3 reduction in energy consumption in the data memory hierarchy on top of the platform independent optimized code. Overall this energy has been reduced with a factor of 16.

8. ACKNOWLEDGEMENTS

This work is partly funded by FWO (Fonds voor Wetenschappelijk Onderzoek).

9. REFERENCES

[1] European Telecommunication Standard ETS 300 401, "Radio broadcasting systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers", *RE/JTC-00DAB-4*, May 1997, Second Edition

[2] *Digital Audio Broadcasting: Principles and Applications*, ISBN 0-471-85894-3, W.Hoeg and T.Lauterbach editors, J.Wiley & Sons Publ., New York, 2001

[3] F.Catthoor, K.Danckaert, C.Kulkarni, E.Brockmeyer, P.G.Kjeldsberg, T.Van Achteren, T.Omnes, *Data access and storage management for embedded programmable processors*, ISBN 0-7923-7689-7, Kluwer Acad. Publ., Boston, 2002

[4] http://research.compaq.com/wrl/people/jouppi/CACTI.html

[5] 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, 2001.

[6] "Trimedia TM1300 Preliminary Data Book", Philips Electronics North America Corporation, 1997.

[7] www.semiconductors.philips.com/trimedia/products/media_proc_ic/

[8] M.Miranda, F.Catthoor, M. Janssen, H.De Man, *High-Level Address Optimization and Synthesis Techniques for Data-Transfer Intensive Applications*, IEEE Trans. on VLSI Systems, no.4, vol.6, Dec. 1998.

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

[10] J.W. Davidson and S. Jinturkar, "Memory Access Coalescing: A Technique for Eliminating Redundant Memory Accesses", in Proceedings of PLDI, June 1994, pp. 186-195.

[11] P.R. Panda, N.D. Dutt and A. Nicolau, "Memory Data Organization for Improved Cache Performance in Embedded Processor Applications", in Design Automation of Electronic Systems, vol. 2, no. 4, pp. 384-409, 1997.

[12] D. Lopez, M. Valero, J. Llosa and E. Ayguade, "Increasing Memory Bandwidth with Wide Buses: Compiler, Hardware and Performance Trade-offs", in Proceedings of ICS-11, July 1997, pp. 12-19.

[13] P. Ellervee, M. Miranda, F. Catthoor and A. Hemani, "System-level Data-format Exploration for Dynamically Allocated Data Structures", in IEEE Trans. on Computer-Aided Design, vol. 20, no. 12, pp. 1469-1472, December 2001.

[14] R. Leupers and S. Bashford, "Graph based Code Selection Techniques for Embedded Processors", in ACM Design Automation of Electronic Systems, vol. 5, no. 4, pp. 794-814, October 2000.

[15] M. Lorenz, D. Kottman, S. Bashford, R. Leupers and P. Marwedel, "Optimized Address Assignment for DSPs with SIMD Memory Accesses", in Proceedings of ASP-DAC, January 2001.

[16] K. Masselos, F. Catthoor, C.E. Goutis, H. De Man, "Interaction between Sub-word Parallelism Exploitation and Low Power Code Transformations for VLIW Multi-media Processors", in IEEE Volta Memorial Workshop on Low-Power Design, March 1999.

[17] S. Julien and N. Drach-Temam, "Memory Bandwidth: The True Bottleneck of SIMD Multimedia Performance on a Superscalar Processor", in Proceedings of EuroPar, August 2001, pp. 439-447.

[18] E. Salamí, J. Corbal, M. Valero, "An Evaluation of Different DLP Alternatives for the Embedded Media Domain", in Proceedings of 1st Workshop on Media Processors and DSPs, November 1999.

[19] J. Huisken, F. van de Laar, M. Bekooij, G. Gielis, P. Gruijters, F. Welten, "A Power-Efficient Single-Chip OFDM Demodulator and Channel Decoder for Multimedia Broadcasting", *IEEE Journal of Solid-State Circuits*, Vol. 33, nr 11, pp.1793-1798, Nov. 1998.

[20] K.Danckaert, F.Catthoor, H.De Man, "A loop transformation approach for combined parallelization and data transfer and storage optimization", *Proc. ACM Conf. on Par. and Dist. Proc. Techniques and Applications*, PDPTA'00, pp.2591-2597, Las Vegas NV, June 2000.