Interaction Between Data Parallel Compilation and Data Transfer and Storage Cost Minimization for Multimedia Applications

Chidamber Kulkarni[1], Koen Danckaert[1], Francky Catthoor[1,2], and Manish Gupta[3]

[1] IMEC, Kapeldreef 75, B-3001 Leuven, Belgium
[2] Professor at the Katholieke Universiteit Leuven
[3] IBM T.J. Watson Research Center, Yorktown Heights, NY

Abstract. Real-time multi-media applications need large processing power and yet require a low-power implementation in an embedded programmable parallel processor context. Our main contribution in this context is the proposal of a formalized DTSE (data transfer and storage exploration) methodology, which allows to significantly reduce system bus load and hence overall system performance and also power consumption. We demonstrate the complementarity of this methodology by coupling the DTSE with a state-of-the-art performance optimizing and parallelizing compiler. Experiments on two real-life video and image processing applications show that this combined approach heavily reduces the memory accesses and bus-loading and hence power and also significantly reduces the total execution time. Decomposing the detailed parallelization and DTSE issues into two different stages is important to obtain the benefits of both the stages without exploding the complexity of solving all the issues simultaneously.

1 Introduction and Related Work

Parallel machines were mainly, if not exclusively, being used in scientific communities until recently. Lately, the rapid growth of real-time multi-media applications have brought new challenges in terms of the required processing (computing) power and power requirements. For this type of applications, especially video, graphics and image processing, the processing power of traditional uniprocessors is no longer sufficient. This has lead to the introduction of small- and medium-scale parallelism in this field too, but then mostly oriented towards single chip systems for cost reasons. Today, many weakly parallel video and multi-media processors are emerging (see [14] and its references), increasing the importance of parallelization techniques. Applications on these processors are parallelized manually even now, which can be tedious and error-prone. This paper presents evidence that parallelizing compilers can be used effectively to deal with this problem, if they are combined with other techniques.

Indeed, the cost functions to be used in these new emerging application fields are no longer purely performance based. Power is also a crucial factor, and has to be optimized for a given throughput. Real time multi-media processing (RMP) applications are usually memory intensive and a significant part of the power consumption is due to the data transfers i.e. in the memory hierarchy [16].

In a parallel processor context most of the research effort in the community so far addresses the problem of parallelization and processor partitioning [2]. Existing approaches do not sufficiently take into account the background storage and transfer related cost. A first approach for more global memory optimization in a parallel processor context was described by us in [6]. Although many software compilers try to come up with the best array layout in memory for optimal cache performance (see e.g. [5] and its references for a few formal approaches based on compile-time analysis) they do not try to directly reduce the storage requirements as memory is allocated based on the available variable declarations. However, in general, this can lead to a large over-allocation, compared to the maximal amount of memory which is really needed over time. We have shown in [4, 9] that aggressive in-place mapping of array signals, based on a detailed life-time analysis, can heavily reduce data storage requirements and improve the cache performance significantly.

The above issues have been a motivation for us to carry out this study. In this paper, we apply our global DTSE (data transfer and storage exploration) methodology on two real-life RMP applications and then couple this to a state-of-the-art optimizing and parallelizing compiler using directives. In the process, we show that the results are very promising (see section 4).

The remaining paper is organized as follows. In section 2, we identify the problem and present the cost functions used in this paper. Section 3 presents the design methodology used in this work. This is followed by discussion and experimental results on two real-life demonstrators in section 4. Conclusions from this work are provided in section 5.

2 Problem Exploration and Cost Functions

In this section we will identify the problems for which we will provide promising solutions. Also the cost functions that we have used in this paper are presented.

As stated in section 1, the emerging real-time multi-media applications demand large processing power and a low-power implementation. Thus in the context of programmable multimedia solutions on parallel processors, the following problems become evident :

1. Do current multi-media applications lend themselves easily to automatic (or directives-based) parallelization?
2. What are the effects of power-oriented program transformations on the performance of (parallel) multi-media applications?

In the sequel, we will provide answers for the above issues. Moreover, we will also show that optimizations during the DTSE stage help in reducing the communication cost (memory accesses) between processor and the (shared) memory.

Two cost functions are used in this paper for evaluating the effectiveness of our methodology. The first one is the execution time of the concerned application. This time is the total execution time of the program measured using UNIX and C time functions. The second cost function is the power consumption. The power consumption is estimated by evaluating the total number of accesses to the off-chip memories. We will calculate both the total number of accesses and size of memories for the demonstrators in section 4.

3 Design Methodology

In this section we present a brief discussion of the main steps in the DTSE stage and the optimizing and parallelizing compiler. Note that the emphasis here is not on the detailed issues which have been presented elsewhere but just to highlight the main benefits of each individual stage.

3.1 Data Transfer and Storage Exploration

The five steps comprising our methodology for system-level power optimization for real-time multi-media applications are illustrated in figure 1. All the steps of our methodology are currently performed manually in this paper but each individual step has been applied systematically in the proposed sequence. Moreover, we are building a prototype compiler supporting these steps. Below is a brief discussion of the different steps in our methodology :

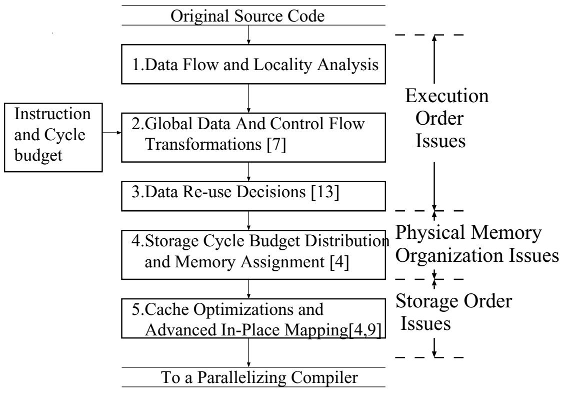
Original Source Code
1.Data Flow and Locality Analysis
Instruction and Cycle budget
2.Global Data And Control Flow Transformations [7]
3.Data Re-use Decisions [13]
4.Storage Cycle Budget Distribution and Memory Assignment [4]
5.Cache Optimizations and Advanced In-Place Mapping[4,9]
To a Parallelizing Compiler
Execution Order Issues
Physical Memory Organization Issues
Storage Order Issues

Fig. 1. Data Transfer and Storage Exploration (DTSE) methodology

The program transformation methodology comprises two phases : in the first phase all the transformations, steps 2, 3, 4 and 5 in figure 1, are chosen based on the amount of reduction in the number of memory accesses to the larger off-chip memories. In the second phase we obtain feedback on the effects of these transformations on performance (delay), static instruction count and code size. Note that these phases are local to each step i.e. the real-time feedback is local to each step. Parts of code exhibiting larger delay are transformed again for improved performance. In addition, in order to reduce the adverse impact of complex conditional expressions and addressing after power oriented transformations, we perform advanced code motion and modulo reductions prior to conventional instruction-level compiler(s) using techniques as described in [12]. Also note that steps 2, 3 and 4 modify the execution order of the application, whereas step 5 only modifies the storage order. This approach has significant benefits as illustrated in [10].

3.2 Performance Optimization and Parallelization

We have used a prototype version of the IBM C compiler (xlc) for AIX 4.2 in our experiments to demonstrate the complementarity to the state-of-the-art parallel compiler. This compiler uses a language-independent optimizer, the Toronto Portable Optimizer (TPO), for global program optimizations and parallelization, and finally, an optimizing back-end for target-specific optimizations. TPO performs (inter-procedurally) classical data flow optimizations like constant propagation, copy propagation, dead code elimination, and loop-invariant code motion [11]. It also performs loop transformations like fusion, unimodular transformations, tiling, and distribution to improve data locality and parallelism detection [18]. Finally, TPO supports parallelization of loops based on both data dependence analysis and user directives. It supports various static and dynamic scheduling policies for nested parallelization of loops [8].

Our experiments have shown that the key analysis needed to detect parallelism automatically in the critical loops of the application was a sophisticated form of array privatization analysis [17]. While TPO does perform inter-procedural array privatization analysis, it failed to detect the privatizability of arrays indexed by expressions involving modulo operations. The analysis of these kinds of array references has not received much attention in the literature, as they do not appear very frequently in scientific programs. The DTSE steps, on the contrary, often introduce modulo arithmetic in the computation of array subscripts in the transformed loops. Even though the compiler was unable to detect the parallelism in many loops automatically, it was quite easy to make it parallelize any loop by simply using a *parallel loop* directive. The compiler automatically privatizes all variables declared inside the loop being parallelized. Finally, since the computations in these applications tend to be quite regular, we used the default, static scheduling policy for scheduling parallel loops.

4 Experimental Results

In this section we address the questions identified in section 2 in the context of implementation of RMP applications on parallel processors.

4.1 Experimental Set-Up

The experimental set-up comprises two main stages (described in section 3): First the concerned application is optimized for data transfer (power) and storage using the DTSE approach. This application is now analyzed for the power and performance cost using the criteria described in section 2. Next the DTSE-optimized application is now fed to the optimizing and parallelizing compiler (OPC). The resulting code from OPC is executed on a 4-way SMP machine (each node is a Power PC 604 with clock frequency of 112 MHz) to observe the performance characteristics of the parallelized code.

Thus we obtain a power estimation of the original and power-optimized code and performance values for DTSE-optimized and OPC-DTSE-optimized code. This allows us to do a fair comparison of the effects of the DTSE stage on performance and parallelization related optimizations.

4.2 Cavity Detection

The cavity (or edge) detection technique is an important step in many (especially medical) image processing applications [3]. This algorithm mainly consists of a sequence of four distinct steps, each of which computes new matrix information from the output of the previous step as shown in figure 2. In this paper, we assume that the image enters and leaves the system in the conventional row-wise fashion.

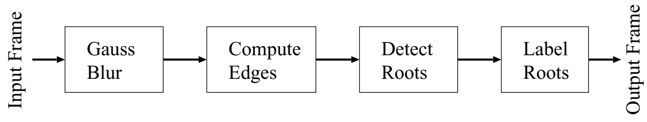

Fig. 2. The Cavity Detection algorithm

Table 1 gives an overview of the reduction of data transfers and the number of memories, accomplished by the different system-level DTSE transformations. The initial algorithm requires 36 accesses per pixel to the frame memory. Whereas the locally optimized algorithm needs 10 accesses to the frame memory and the globally optimized algorithm needs at most two accesses to the input/output buffers. Thus with DTSE transformations, a significant reduction in power and area is obtained for this example. Table 3 gives the execution times for different optimization levels for the parallelized code on different number of processors. Here we observe that the total execution time decreases consistently after the DTSE optimizations, and the optimized version performs better for larger number of processors. Thus apart from reducing the power cost, our DTSE approach heavily enhances the performance on parallel machines for data dominated applications by reducing the potential inter-processor communication.

Version	Parallelism	# Main memory	# Frame transfers
Initial	data	2 (+1)	36
	task	8 (+1)	36
Locally Optimized	data	1 (+1)	10
	task	4 (+1)	10
DTSE optimized	data	0	0(2)
	task	0	0(2)

Table 1. Data transfer and main memory storage requirements of different cavity detection mapping alternatives on a parallel processor. If the input frame buffers are included, the numbers between () are obtained. Frame transfers are counted per pixel.

4.3 Quad-Tree Structured Differential Pulse Code Modulation (QSDPCM)

The QSDPCM [15] technique is an interframe adaptive coding technique for video signals. A global view of the QSDPCM coding algorithm is given in figure 3. Most of the submodules of the algorithm operate in cycles with an iteration period of one frame (see table 3 for frame sizes). It is worth noting that the algorithm spans 20 pages of complex C code. Hence it is practically impossible to present all the detailed dependencies and profile of the algorithm. A more detailed explanation of the different parts of the algorithm is available in [15]. Table 2, gives the number (and size) of memories required and the correspond-

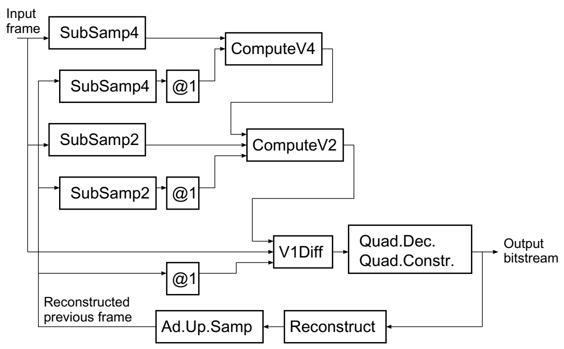

Fig. 3. The QSDPCM algorithm

ing number of accesses for the QSDPCM algorithm. We observe that there is a significant reduction in the total number of off-chip accesses for both the memories. Hence the total power consumption, for the global DTSE-transformed case is heavily reduced compared to the original. Table 3 shows that the DTSE-transformed parallelized code has better execution times compared to that of the original parallelized code. Note that the execution time for the original case reduces by a factor three with four processors, whereas for the DTSE transformed case there is even a reduction in execution time by an almost ideal factor 3.9. This once again shows that DTSE is complementary and is even augmenting the effect of the more conventional performance and parallelization related optimizations.

Version	Memory Size	# Accesses
Initial	206K	5020K
	282K	8610K
DTSE	206K	4900K
transformed	1334	8548K

Table 2. Amount of memory and the corresponding total number of accesses to each of these memories for the QSDPCM algorithm for different optimization levels.The memory size is in bytes.

Version	Frame Size	P=1	P=2	P=3	P=4
Cavity Detection					
Original	640 × 400	0.87	0.65	0.63	0.63
	1280 × 1000	4.47	3.19	3.01	3.06
	12800 × 10	4.07	3.00	2.94	3.03
DTSE	640 × 400	0.32	0.27	0.21	0.18
transformed	1280 × 1000	1.71	1.30	0.98	0.83
	12800 × 10	3.11	1.92	1.79	1.61
QSDPCM					
Original	288 × 528	5.89	3.11	2.26	2.00
	576 × 1056	22.83	12.32	9.09	7.62
DTSE	288 × 528	3.48	1.75	1.18	0.98
transformed	576 × 1056	13.45	7.28	4.79	3.47

Table 3. Performance of parallelized Cavity detection and QSDPCM algorithm for various cases using a Power PC 604 based 4-way SMP machine. Here P represents the number of processors and the execution time is in seconds.

4.4 Summary

Below is a summary of the main results from the experiments with the two real-life applications :

1. We are able to parallelize both the codes using manual directives to guide the parallelizing compiler.
2. The performance results for both the demonstrators show that DTSE transformations are complimentary to and even augmenting the effect of the performance related optimizations (a gain between factor 2.5 and 3.9 for four

processors).This comes on top of the power and bus load reduction effects of the DTSE step.

3. Automatic parallelization of these multimedia demonstrators is possible but requires a sophisticated array privatization scheme (see section 3.2). Modulo operations introduced by DTSE transformations do not effect the inherent parallelism in the code but requires a more complex analysis to detect the parallel loops. Modulo reduction as described in [12] has large benefits. Moreover, using manual directives it is very well possible to obtain large speed-ups in execution times as illustrated in the results.

5 Conclusions

The main conclusions of this paper are : (1) System bus-load and power optimizing DTSE program transformations are complementary to the pure performance related program transformations in data parallel compilers and (2) DTSE transformations do not effect the inherent parallelism present in an application and even enhance the performance itself by reducing the inter-processor communication. So our DTSE approach is fully complementary to the conventional performance related design approaches, and combining the two approaches has major benefits. This is demonstrated on two real-life parallel multimedia demonstrators which have been mapped on a IBM 4-way SMP machine.

References

[1] A.Agarwal, D.Krantz, V.Nataranjan, "Automatic partitioning of parallel loops and data arrays for distributed shared-memory multiprocessors", *IEEE Trans. on Parallel and Distributed Systems*, Vol.6, No.9, pp.943-962, Sep. 1995.

[2] U.Banerjee, R.Eigenmann, A.Nicolau, D.Padua, "Automatic program parallelization", *Proc. of the IEEE*, invited paper, Vol.81, No.2, Feb. 1993.

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

[4] F.Catthoor, S.Wuytack, E.De Greef, F.Balasa, L.Nachtergaele, A.Vandecappelle, "Custom Memory Management Methodology – Exploration of Memory Organization for Embedded Multimedia System Design", ISBN 0-7923-8288-9, Kluwer Acad. Publ., Boston, 1998.

[5] M.Cierniak, W.Li, "Unifying Data and Control Transformations for Distributed Shared-Memory Machines", *Proc. of the SIGPLAN'95 Conf. on Programming Language Design and Implementation*, La Jolla, pp.205-217, Feb. 1995.

[6] K.Danckaert, F.Catthoor and H.De Man, "System-level memory management for weakly parallel image processing", *In proc. EUROPAR-96*, Lecture notes in computer science series, vol. 1124, Lyon, Aug 1996.

[7] E.De Greef, F.Catthoor, H.De Man, "Program transformation strategies for reduced power and memory size in pseudo-regular multimedia applications", accepted for publication in *IEEE Trans. on Circuits and Systems for Video Technology*, 1998.

[8] S. Hummel and E. Schoenberg, "Low-overhead scheduling of nested parallelism",*IBM Journal of Research and Development*, 1991.

[9] C.Kulkarni, F.Catthoor, H.De Man, "Hardware cache optimization for parallel multimedia applications", *In Proc. of EuroPar'98*, Southampton, pp. 923-931, Sept 1998.

[10] C.Kulkarni, D.Moolenaar, L.Nachtergaele, F.Catthoor, H.De Man, "System-level energy-delay exploration for multi-media applications on embedded cores with hardware caches", Accepted for *Journal of VLSI Signal Processing*, special issue on SIPS'97, No.19, Kluwer, Boston, pp., 1999.

[11] S. Muchnick, "Advanced compiler design and implementation", *Morgan Kaufmann Publishers Inc.* , ISBN 1-55860-320-4, 1997.

[12] 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*, Vol.7, No.1, March 1999.

[13] J.Ph. Diguet, S. Wuytack, F.Catthoor, H.De Man, "Formalized methodology for data reuse exploration in hierarchical memory mappings", *In Proc. Int'l symposium on low power electronics and design*, pp.30-36, Monterey, Ca., Aug 1997.

[14] P.Pirsch, H-J.Stolberg, Y-K.Chen, S.Y.Kung, "Implementation of Media Processors", *IEEE Signal Processing Magazine*, No.4, pp.48-51, July 1997.

[15] P.Strobach, "QSDPCM – A New Technique in Scene Adaptive Coding," *Proc. 4th Eur. Signal Processing Conf.*, EUSIPCO-88, Grenoble, France, Elsevier Publ., Amsterdam, pp.1141–1144, Sep. 1988.

[16] V.Tiwari, S.Malik and A.Wolfe, "Instruction level power analysis and optimization of software", *Journal of VLSI signal processing systems*, vol. 13, pp.223-238, 1996.

[17] P. Tu and D. Padua, "Automatic array privatization", *Proc. 6th Workshop on Languages and Compilers for Parallel Computing*, Portland, OR, August 1993.

[18] M. J. Wolfe, "Optimizing Supercompilers for Supercomputers",*The MIT Press*, 1989.