

# Data Cache Optimization in Multimedia Applications

A.M. Molnos<sup>(\*)(\*\*)</sup>, M.J.M. Heijligers<sup>(\*\*)</sup>, S.D. Cotofana<sup>(\*)</sup>,

J.T.J. van Eijndhoven<sup>(\*\*)</sup>, B. Mesman<sup>(\*\*)</sup>

<sup>(\*)</sup> Delft University of Technology

Mekelweg 4, 2628 CD, Delft, The Netherlands

Phone: 015-2786196 Fax: 015-2784898

email: {ancutza, sorin}@dutepp0.et.tudelft.nl

<sup>(\*\*)</sup> Philips Research Laboratories

Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands

email: {marc.heijligers, jos.van.eijndhoven, bart.mesman}@philips.com

*Abstract*—

Cache memories are a widely used approach to fill the processor-memory speed gap such that the bandwidth requirements of today's data intensive multimedia applications can be achieved. The performance of the system is strongly related with the performance of the memory hierarchy.

In this paper we address the problem of improving performance of the memory hierarchy at system level for multitasking data intensive applications. Our proposed new method separates the problems of intra task data cache misses and inter-task data communication cache misses by using cache partitioning.

For the intra-task data cache misses compiler-like methods can be used for optimization.

For inter-task data communication we propose a new analytical method to find a static task execution order that optimizes the number of inter-task data cache misses for a single processor architecture. Our analytical method is based on solving the Integer Linear formulation of the problem.

*Keywords*— multimedia applications, shared data cache, task scheduling

## I. INTRODUCTION

To cope with the increase in complexity and size of signal-processing applications, more computational power is required in order to fulfil the real time constraints of their implementation.

The increase in computational power implies more data traffic from computational units to storage. The increase in bandwidth to off-chip memories is not growing as fast as the increase in speed of computation power, leading to a 50% processor/memory speed gap, as shown in [4]. An approach to deal with memory bandwidth bottlenecks is to use memory hierarchies (caches) [4]

In state of the art system design data cache misses optimizations are usually done at compiler level. For applications that exhibit task level parallelism there is additional

freedom for optimization. The task schedule gives the moment and order of productions and consumptions of communicated data. Depending on these access orders and times different performances of the memory system measured in number of cache misses can be obtained.

This paper proposes a new method that tackles the general problem of minimizing data cache misses at system level for parallel data intensive applications. We separate this general problem into two parts corresponding to the inside task cache conflicts and the inter-task communication cache conflicts. This decoupling is done by using a cache partitioning method that assigns different exclusive cache parts to tasks and one share cache pool for the communication data. In [8] a task level cache partitioning method is presented and can be easily extended with a share cache pool for the communication data.

For the intra-task case compiler methods like the ones for the sequential applications can be used to improve data cache behavior [4].

For inter-task communication we present a new analytical method to find a static task execution order for a single processor architecture such that the number of inter-task data misses is optimized, given are the cache parameters and for every communication data the start address (as set number) and the size in cache.

The outline of this paper is as follows. In section II related work is presented, in III a short overview on potential optimization opportunities for multimedia applications and our new optimization general method for parallel applications are presented, then in section IV we propose our new method to find a static task execution order. In section V the experiments done for our method are presented and finally the conclusions and lines for future research are drawn in section VI.

## II. RELATED WORK

In the problem of optimizing data cache misses for the sequential applications case a lot of research has been done. For the case of multidimensional arrays a global approach is taken in [2]. In [2] both of the problems of improving spacial locality (by proper array address assignment) and improving temporal locality (by proper instruction scheduling) are tackled. For the optimization, all operations and loop nestings are taken into account but the approach is restricted to sequential applications.

For parallel applications in [9] a thread scheduling method that improves cache locality is presented. The only case tackled there is the case of threads without data dependencies.

## III. OVERVIEW

In this section we present the ingredients used to build our method of minimizing data cache misses at system level. We identify two types of multimedia data intensive applications: applications that have explicit parallelism at task level (task level parallel applications, figure 1) and applications where data dependencies impose that tasks are executed in a sequential way (sequential applications, figure 2).

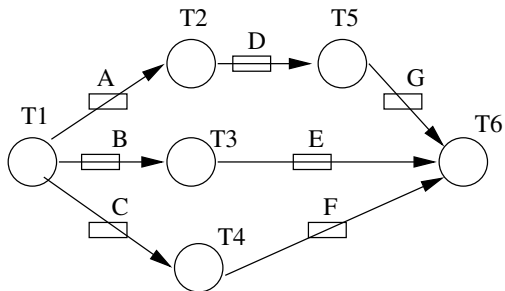


Fig. 1. Parallel application example. Tasks: T1-T6; Communication buffers: A-G

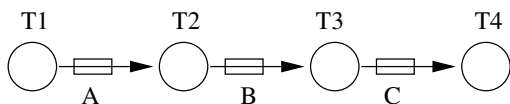


Fig. 2. Sequential application example. Tasks: T1-T4; Communication buffers: A-C

In the followings the methods for optimizing data cache misses corresponding to the two types of applications are presented.

### A. Sequential applications

For sequential applications we rely on work already published in literature. These optimizations are compiler-

based and are done by pipelining, parallelising and changing execution order of the code at instruction level. For this optimizations data flow analysis and lifetime of variables analysis are suitable [3] [2].

Improvements can be obtained using software techniques which reorder the program loops structure in order to increase spatial data locality and reuse [7], [1], [5] and data layout techniques which change the memory layout of multi-dimensional arrays and data structures from the default specification (for example row major for C) such that the spatial locality is improved [6]. Usually these types of optimizations are performed simultaneously.

Techniques that do aggressive loop pipelining and schedules productions as close as possible to consumptions can be used as well at this level of sequential applications [3]. In [3] loop pipelining is actually done implicitly because of the 'as soon as possible' used schedule.

### B. Task level parallel applications

For the parallel applications besides the misses caused by task's internal execution path (called intrinsic misses in [10]) there are extra misses caused by inter task cache interference (called extrinsic misses in [10]). This last type of interference is depending not only on task's behavior but also on task scheduling. At this level our new method comes into place.

Using partitioning every task gets an exclusive cache part for its private data [8], Because of this cache partitioning the intrinsic misses are 'isolated' for every task and same type of optimizations as in III-A can be applied.

A distinct cache partition is assigned for shared inter-task communication data. The extrinsic misses problem is reduced to the interference between shared communication data. At this level the cache optimization is done at task scheduling level as will be presented in details in section IV.

## IV. ANALYTICAL METHOD

In this section we present the analytical method used to optimize the inter-task data cache misses.

We consider the application described as a directed acyclic graph  $G(V, E)$  where the nodes  $V$  represent the tasks of the system and the edges  $E$  represent the data dependencies between tasks and are oriented from the node that produces the data to the node that consumes it.. Every edge  $e \in E$  is augmented with the start address in cache and with the size of the correspondent communication buffer. Let  $n$  be the number of nodes and let  $m$  be the number of edges of the graph  $G$ .

Given the cache parameters and the graph  $G$  describing the application the problem is to find the starting times  $t(i)$

(with  $i = 0..n - 1$ ) for the  $n$  tasks of the system such that the inter task data dependencies constraints are satisfied and the number of cache misses at communication buffers level is optimized. We formulate this problem as an Integer Linear Problem (ILP).

The execution time of every task can be considered to be equal to 1 because the task schedule is assumed non-preemptive. Every communication data is assumed to be produced once and consumed once during one execution of the producer and consumer tasks (this model corresponds to FIFO communication).

In order to express the execution order of the tasks in ILP we use a  $n \times n$  precedence matrix  $P$  where every element:

$$p_{i,j} = \begin{cases} 1 & \text{if node } i \text{ is executed before node } j \\ 0 & \text{otherwise} \end{cases}$$

For every two nodes  $i$  and  $j$   $p_{i,j} = 1 - p_{j,i}$ , because the graph is acyclic. If in the graph  $G$  there is a path between nodes  $i$  and  $j$  then  $p_{i,j} = 1$  and it will not be considered as a decision variable for our ILP formulation.

After the precedence matrix elements are computed the starting times for the tasks are easily deduced.

The cache conflicts between edges are described by the  $m \times m$  reload matrix  $R$ , where every element:

$$r_{e,f} = \begin{cases} 1 & \text{if edge } e \text{ causes reload for edge } f \\ 0 & \text{otherwise} \end{cases}$$

For every edge  $e$  we define two functions *source* and *sink*, that give the node that produces respectively consumes the data corresponding to the edge  $e$ :

$$source(e) : [0, m - 1] \cap \mathbb{N} \rightarrow [0, n - 1] \cap \mathbb{N}$$

$$sink(e) : [0, m - 1] \cap \mathbb{N} \rightarrow [0, n - 1] \cap \mathbb{N}$$

A cache conflict causing misses between two edges exists if the correspondent communication buffers have overlapping address in the cache and their lifetimes overlap.

Temporally it is possible that an edge  $e$  can cause reload for the edge  $f$  if  $e$ 's corresponding data is accessed between the production and the consumption of  $f$ 's corresponding data.

$$r_{e,f} = 1 \Leftrightarrow \begin{aligned} &t_{source(f)} < t_{source(e)} < t_{sink(f)} \text{ or} \\ &t_{source(f)} < t_{sink(e)} < t_{sink(f)} \end{aligned}$$

Using the precedence matrix, if an edge  $e$  causes reload to the edge  $f$  then at least one of the following properties hold:

- the source of  $f$  precedes the source of  $e$  and the source of  $e$  precedes the sink of  $f$

$$r_{e,f} \geq p_{source(f),source(e)} + p_{source(e),sink(f)} - 1$$

- the source of  $f$  precedes the sink of  $e$  and the sink of  $e$  precedes the sink of  $f$ :

$$r_{e,f} \geq p_{source(f),sink(e)} + p_{sink(e),sink(f)} - 1$$

By means of the example in figure 1 we show how the cost function represented by the number of misses is computed. In the graph in figure 1 the edges  $D$ ,  $E$ ,  $F$  can conflict in the cache. Let's assume that the image in the shared data communication cache of the three edges is the one in figure 3.

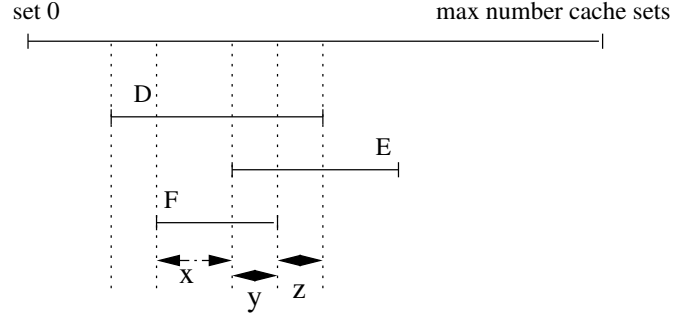


Fig. 3. Placement of edges  $D$ ,  $E$  and  $F$  in the cache space

From the point of view of edge  $D$  in interval  $x$  it can be flushed out of the cache by edge  $F$ , in interval  $y$  it can be flushed out of the cache by edges  $E$  and  $F$  and in interval  $z$  it can be flushed out of the cache by edge  $E$ .

For every edge the cache part occupied by its data is divided in a number of parts equal with the different number of cache intervals in which the edge can interfere with other edges. The cost for every edge is the sum of its costs for the cache intervals defined as previously.

If the cache is direct mapped, for our example the cost for the edge  $D$  is defined as follows:

$$cost(D) = x * r_{E,D} + y * max(r_{E,D}, r_{F,D}) + z * r_{E,D}$$

For the general case of  $N$  ways associative cache the cost for an edge  $e$  in a cache interval  $x$  where  $e$  overlaps with other edges  $f_k$  ( $k = 1..M$ ) is

$$r_{e,x} = \begin{cases} 1 & \text{if } \sum_{k=1}^M r_{e,f_k} - N > 0 \\ 0 & \text{otherwise} \end{cases}$$

Using two extra variables,  $r_{e,x}$  can be linearly defined.

The overall cost of an solution is the sum of the costs for every edge of the graph and it is the objective to be minimized when computing the precedence matrix  $P$ .

## V. EXPERIMENTS

The orthogonal separation of the general problem of cache misses for parallel communicating applications into intra- and inter- task data misses problems made separate reasoning of the performance possible for the two cases.

The intra-task cache performance depends on compiler optimization techniques used at that level.

To test the effectiveness of our inter-task scheduling method a number of examples were made. Randomly generated graphs of tasks were studied to test the robustness of the method. The task execution order found is optimal because the underlying mathematical model is based on ILP. For up to twenty nodes the searching time is in order of magnitude of seconds and for hundred of nodes the searching time is in order of magnitude of tens of minutes. The execution time for the solution search depends heavily also on the number of edges in the task graph (more edges showing dependences - less decision variables for the execution order of tasks).

Our method abstracts from all operation level details that has to be modelled for the methods as in III-A so it can be applied for big sets of tasks, independent of their size.

## VI. CONCLUSIONS

For the problem of optimizing cache behavior for parallel applications new task scheduling techniques are required. We proposed a new general method for improving the cache performance for parallel applications. Our method allows use of others existing compiler's cache optimization methods for the intra-task case and optimizes the inter-task communication misses by changing the schedule. The task execution order found using our method is optimal with respect to number of cache misses at inter-task communication data level.

First experiments showed the viability of the new data cache misses optimization approach for the applications that exhibit enough freedom of reordering task execution. Anyway for existing multimedia applications compiler methods for sequential code case can bring more gain in optimize the caches because of lack freedom in reordering task execution.

Future work will be oriented towards rewriting techniques that improve freedom of reordering task execution. Preemptive scheduling techniques will also be studied because they can bring more freedom for optimization in the execution order of the application.

## REFERENCES

- [1] D. Bacon, S. Graham, and O. Sharp. Compiler transformations for high performance computing. *ACM Computing Survey*, pages 345–420, 1994.
- [2] R. Clout. Periodic scheduling for cache-miss minimisation. *PhD Thesis, Eindhoven University of Technology*, 2001.
- [3] M. J. M. Heijligers and A. Hogenhuis. Analysing architectural aspects of behavioral descriptions. *Conf. on Embedded System Design*, pages 239–250, 2000.
- [4] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, San Fransisco, CA, 2003.
- [5] C.-H. Hsu and U. Kremer. A stable and efficient loop tiling algorithm. *Mid-Atlantic Student Workshop on Programming Languages and Systems, Newark, DE*, 2000.
- [6] N. Manjikian and A. T. Array data layout for the reduction of cache conflicts. *In Proceedings of the 8th International Conference on Parallel and Distributed Computing Systems*, 1995.
- [7] K. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems*, pages 424–453, July 1996.
- [8] A. Molnos, M. Heijligers, J. v. Eijndhoven, and S. Coto-fana. Cache partitioning with guaranteed performance. *Submitted for Design Automation and Test in Europe, Paris*, 2004.
- [9] J. Philbin, J. Edler, O. Anshus, C. Douglas, and K. Li. Thread scheduling for cache locality. *In Proc. of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA*, October 1996.
- [10] F. Sebek. The state of the art in cache memories and real-time systems. *Technical Report MRTC, (01/37)*, Oct. 2 2001.