Cache Partitioning Options for Compositional Multimedia Applications

A.M. Molnos(*)(**), M.J.M. Heijligers(**), S.D. Cotofana(*),
J.T.J. van Eijndhoven(**)
(*) Delft University of Technology
Mekelweg 4, 2628 CD, Delft, The Netherlands
Phone: 015-2786196 Fax: 015-2784898
email:{ancutza, sorin}@dutepp0.et.tudelft.nl
(**) Philips Research Laboratories
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
email:{marc.heijligers, jos.van.eijndhoven}@philips.com

Abstract—

The use of conventional probabilistic cache models in embedded multimedia systems is restricted because the tasks flush each others data out of the cache in an unpredictable manner making the system not compositional. In this way the overall performance is difficult to predict and the integration of new tasks expensive. In a multiprocessor platform the unified levels of cache shared between processors are causing the largest task conflicts. Our proposal is to introduce a compositionality property to the system such that the overall performance can be easily computed if the components' performance in known. This property is obtained by means of assigning each task their exclusive part of the unified level of cache shared between processors. This article also presents the results of the effect on performance (measured in number of cache misses) of two types of cache partitioning, namely set based and associativity based partitioning, for the case of multimedia applications. Our experiments indicate that when applied to a multiprocessor with memory hierarchy the set based partitioning performs always better that the associativity based partitioning. Moreover set based partitioning typically results in performance gain of up to 35% less misses, where the associativity based partitioning always results in large performance degradation of 17% to 150% more misses, when compared with the shared cache case.

I. Introduction

The increase in size and complexity of state-of-the-art multimedia applications require high performance hardware platforms with large memory bandwidth. 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 [10]. A possible approach to cope with this processor-memory gap is to use (distributed) memory hierarchies (caches), such as reported in [10].

In parallel environments caches induce undesired unpredictability because tasks can influence each others performance by flushing each others data out of the cache. Under these conditions it is difficult to guarantee memory hierarchy performance for single tasks, and hence for the system as a whole. In the real-time domain one could think of replacing caches with scratch pad memories, but in practice removing caches is not a good choice because they provide flexibility. They don't need to be resized or modified if the application changes. Detailed analysis regarding the memory requirements of multimedia applications' code and data at every change of standards or features takes much effort and negatively affects the time-to-market.

Our proposal is based on the observation that the following holds true: in a multiprocessor system, the levels of unified cache shared between processors are the most affected by the inter-task conflicts. Thus we propose to achieve compositionality by exclusively assigning parts of the last level of cache to tasks. The problem addressed in this article can be stated as follows: given the tasks that are running concurrently in a composed system, find a cache partitioning strategy that optimizes the overall system performance.

The performance implication of two cache partitioning methods, set-based partitioning and associativity-based partitioning are compared for multimedia workloads, using a multiprocessor CAKE platform [1]. The simulation indicate that, with respect to the number of misses, set-based partitioning always performs better that associativity-based partitioning. Compared with the shared cache case, set-based partitioning can a bring performance improvement of up to 35% less cache misses. In the most disadvantageous studied case this partitioning type results in 27% increase in number of misses. Com-

pared with the shared cache case, associativity-based partitioning brings large performance degradation of 17% to 150% more cache misses.

The outline of this paper is as follows. Section II presents the work carried out till now in this field of inter task cache interference. Section III briefly describes cache partitioning, and proposes two candidates for experiments. Section IV presents the target architecture, the method used to compare cache partitioning types and the practical results. Finally in Section V conclusions are drawn.

II. Related work

In the single processor domain different cache management methods were proposed.

In [3], a hardware method to divide a fully associative cache into partitions for each real-time task and a larger partition called the shared pool for the non-real-time tasks is described.

A software cache partitioning method is presented in [6]. The address space is partitioned among caches at compile and link time. This strategy is applicable only for level one caches.

In [5] an operating system controlled cache partitioning method is proposed. The major drawbacks of this method are that it is limited to physically indexed caches and it limits to one memory page the basic partitioning unit assignable to a task.

Different analytical cache models to estimate the overall miss rate in a multi-threading system are proposed in [8] and [11]. Both of the methods are applicable to fully associative caches with a last recently used replacement strategy (set associative caches are approximated as fully associative). Only the case of column caching is studied.

In [9] the problem of cache efficiency for simultaneous threads in a time-sharing environment is tackled using dynamic column caching, in a best-effort way. Based on their number of misses tasks are dynamically ”stealing” eachother cache ways, such that the overall number of misses is improved. In [11] the problem of optimal allocation of cache between two competing processes that minimizes the overall miss rate is presented.

A task allocation scheme for multi-rate systems on multiprocessor based on a cache partitioning and reservation technique is presented in [4]. The fact that only the instructions caches are modeled makes this method inapplicable for preserving the individual task performance in the case of unified shared caches.

In [7] is introduced a compositional cache model that used specialized cache management instructions. This strategy is applicable only for level one data caches.

The main differences between our work and the previous research is that we tackle the case of applications running on platforms with unified set-associative cache shared between the processors. For multimedia applications the impact of different types of cache partitioning is studied.

III. Cache partitioning

The platform model for the proposed method is a homogeneous on-chip multiprocessor having high bandwidth communication network with the partitionable shared unified on-chip cache. On this platform applications consisting of parallel tasks are executed. The underlying idea of the proposed partitioning strategy is to give different amount of cache to the tasks.

In a conventional set associative cache organization the address is split into three parts: tag, index and offset [10]. As can be seen also in the Figures 1 and 2 the index directly addresses the cache set in which the data might be. The tag part of the address is compared against the tag part stored in the cache to determine if there is a hit or a miss. The offset part of the address selects the desired word in the cache block.

With respect to conventional cache organization we identify two main possible types of partitioning:

- Based on associativity (Figure 1) - tasks get a number of ways from every set of the cache. At this level the granularity of the partitioning is limited to the number of ways in a set (cache organization). This partitioning type is commonly used because the implementation doesn't require modifications in the structure of the cache (tag, index size) or in the addressing mode. Implementing it requires a small change in the hardware responsible with the replacement policy such that, depending on the task that accesses the memory, different ways are searched.
- Based on sets (Figure 2) - tasks get different amount of sets from the cache. Depending on the desired partition granularity, a number of bits from the index part of the address should be translated such that the accessed memory location goes into the allocated cache partition. The implementation requires the hardware support for address translation and allows a finer granularity than partitioning based on associativity.

A mixed set and associativity based partitioning is possible but both implementation overhead will add, so the cache will become too slow.

Both presented types of cache partitioning are suitable for solving the predictability problem of a system consisting in multiple tasks sharing the cache. The granularity of the partitioning should be fine enough in order to be able to allocate different cache amount for each task. From this

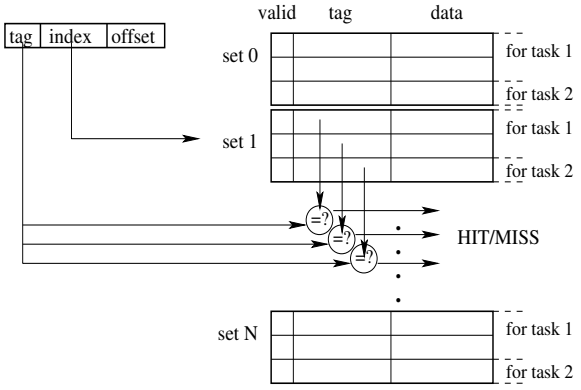

Fig. 1. Associativity-based cache partitioning

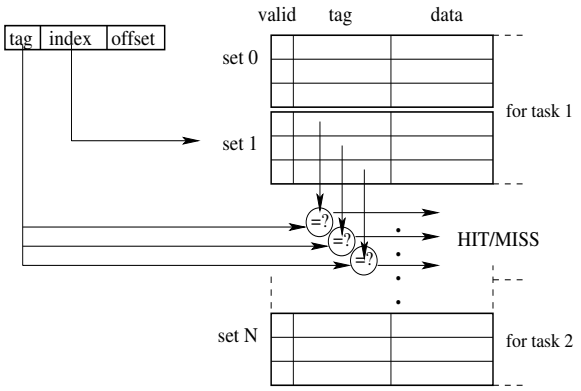

Fig. 2. Set-based cache partitioning

point of view set based partitioning is preferred.

In the next section we will compare the performance of the two types of partitioning measured in number of misses.

IV. Framework

The used applications consist of parallel tasks. The performance (measured in number of misses) comparison between different cases of cache partitions and the shared cache case is made. Due to compositionality property introduces by the cache partitioning every task's number of misses can be obtained by simulating that task in isolation, on a single-processor having different cache parameters. For the shared cache case all the task are simulated concurrently on a multiprocessor.

The operating system can be treated as a separate task with its own part of the cache. In our examples, its cache interference was only present when accessing data structures for the file system and therefore it is negligible.

A. Target architecture

Our target architecture is a multi-processing architecture (for increased performance), together with distributed memory hierarchies (for increased bandwidth). For our experiments we use a practical instance of such a target architecture, called the CAKE platform [1].

The CAKE platform consist of a homogeneous network of computing tiles on a chip. Each tile contains CPUs (trimedia and/or MIPS), a router (for out-tile communication) and memory banks (Figure 3). The processors are connected to memory by a fast high-bandwidth snooping interconnection network. The on tile memory is actually used as a level 2 cache, shared between tasks, facilitating a fast access to the main memory which is outside chip.

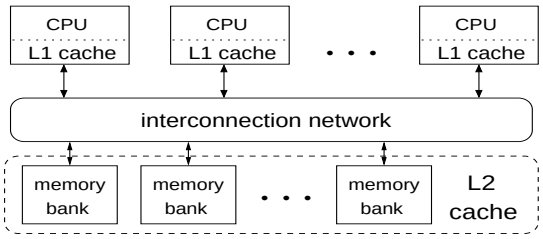

Fig. 3. CAKE architecture - tile view

For our experiment, we used the level 2 cache of CAKE platform which is shared between tasks to study the impact of cache partitioning type on performance of the system. The possible allocated cache size for a task is limited to powers of two times the associativity of the cache.

B. Practical results

The application workloads for the studies are belonging to MediaBench [2]. The four examples of applications presented here are as follows: a picture in picture (PiP) application consisting of two concurrent mpeg2 decoders working on different video streams, a PiP application consisting of three concurrent mpeg2 decoders working on different video streams, a mpeg2 encoder, jpeg decoder and mpeg2 decoder running in parallel and a mpeg2 encoder, epic decoder, adcpm decoder and h26l encoder part running in parallel.

In the first example the level 2 cache has a size of 1MB, is 8-ways associative, 256 sets and is partitioned in three different ways. The studied cases are:

- *shared:* the two mpeg2 decoders are sharing the complete L2 cache.
- *set-based partitioning:* every decoder task gets 128 sets of the cache.
- *assoc-based partitioning1:* every decoder task gets 4 ways of every cache set.
- *assoc-based partitioning2:* the decoder task that works on the smaller picture gets 3 ways of the cache and the other decoder gets 5 ways of the cache.

The corespondent number of misses for every cache configuration are presented in Figure 4. In terms of cache

misses it was observed that partitioning at associativity level (*assoc-based partitioning1, assoc-based partitioning2*) gives a performance degradation of at least 17% respectivelly 35% compared to the performance of the shared cache. For the set level partitioning the performance is increased by 13%.

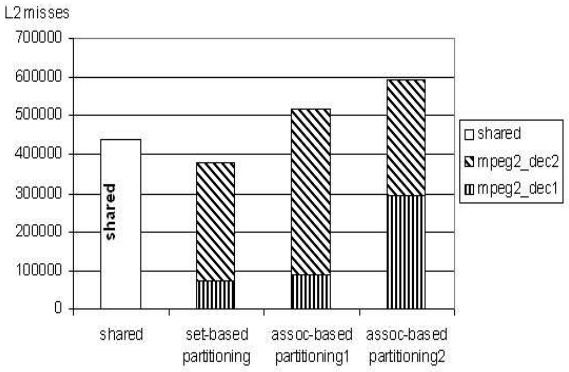

Fig. 4. Cache misses for different partition types - example 1

In the second example the level 2 cache has a size of 1MB, is 8 ways associative, has 256 sets and was partitioned in four different ways. The studied cases are:

- *shared:* the three decoder tasks are sharing the complete L2 cache.
- *set-based partitioning:* the decoder task that works on the largest picture gets 128 sets and each of the other two decoders get 64 sets.
- *assoc-based partitioning1:* the decoder task that works on the largest picture gets half of the cache ways (4 ways) and each of the other two decoders get a quarter of the cache's ways (2 ways).
- *assoc-based partitioning2:* two decoders get each 3 ways from every set and the other only 2 ways.
- *assoc-based partitioning3:* the same ratio as in case 4, but a different allocation between tasks.

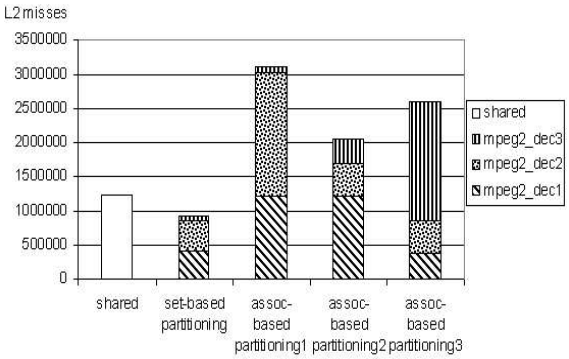

Fig. 5. Cache misses for different partition types - example 2

The obtained corespondent number of misses for the five cache configuration of the second example are presented in Figure 5. In terms of cache misses it was observed that partitioning at associativity level gives a performance degradation of 152% (*assoc-based partitioning1*), 66% (*assoc-based partitioning2*) and 110% (*assoc-based partitioning3*). In this example the L2 misses are decreased with 24% in the case of set level partitioning.

In the third example the level 2 cache has a size of 1MB, is 8 ways associative, has 256 sets and was partitioned in three different ways. The studied cases are:

- *shared:* tasks are sharing the complete L2 cache.
- *set-based partitioning:* the mpeg2 decoder gets 128 sets and the jpeg decoder and the mpeg2 encoder get every one 64 sets.
- *assoc-based partitioning1:* the mpeg2 decoder has 4 sways of every set and the jpeg decoder and mpeg2 encoder have everyone 2 ways of every set.
- *assoc-based partitioning2:* the mpeg2 decoder and the mpeg2 encoder have everyone 3 ways of every set and the jpeg decoder has 2 ways of every set.

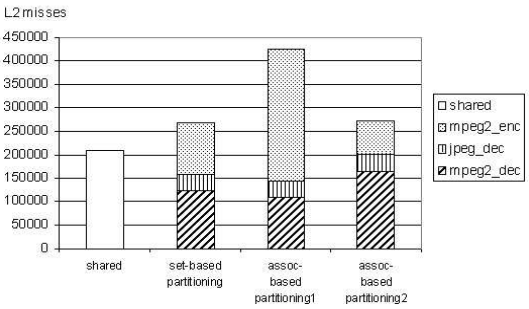

Fig. 6. Cache misses for different partition types - example 3

The obtained corespondent number of misses for the four cache configuration of the third example are presented in Figure 6. In terms of cache misses it was observed that all the partitioning ratio simulated give a performance degradation (27% the *set-based partitioning*, 102% the *assoc-based partitioning1* and 30% (*assoc-based partitioning2*).

In the forth example the level 2 cache has a size of 2MB, is 8-ways associative, has 512 sets and it was partitioned in two different ways. The simulated cases are:

- *shared:* the tasks are sharing the complete L2 cache.
- *set-based partitioning:* every task gets a forth of the number of the cache sets.
- *assoc-based partitioning1:* every task gets 2 ways of every cache set.

The corespondent number of misses for every forth example cache configuration are presented in Figure 7. In terms of cache misses it was observed that partitioning at associativity level (*assoc-based partitioning1*) gives a performance improvement of 1% and the set-based partitioning

gives a performance improvement of 30% compared to the performance of the shared cache.

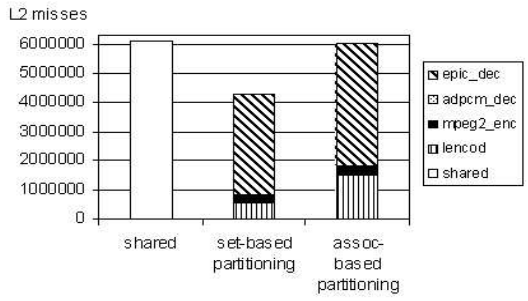

Fig. 7. Cache misses for different partition types - example 4

V. Conclusions

Due to inter-task cache conflicts shared caches cause undesired unpredictability when used in a embedded multiprocessor environment. To eliminate inter-task conflicts and induce compositionality to the system this article propose partitioning the unified level of cache shared between processors. Two options of cache partitioning are available: set based partitioning and associativity based partitioning. Studying memory performance of multimedia workloads in multiprocessor environments, we found that set-based partitioning is always better that associativity-based partitioning in terms of number of misses. Compared with the shared cache case, set-based partitioning can bring performance improvement up to 35% less misses. In the most defavorable studied case this partitioning type results in 27% increase in number of misses. Compared with the shared cache case, associativity-based partitioning brings large performance degradation of 17% to 150% more cache misses.

Future research will be oriented toward finding the partitioning ratio such that the best performance gain for the overall system is obtained.

References

[1] P. Stravers, J. Hoogerbrugge. Homogeneous Multiprocessing and the Future of Silicon Design Paradigms In *Proceedings, International Symposium on VLSI Technology, Systems, and Applications (VLSI-TSA)* 2001.

[2] L. Chunho, M. Potkonjak, W.H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *International Symposium on Microarchitecture*, 1997.

[3] D.B. Kirk. SMART (Strategic Memory Allocation for Real-Time) Cache Design. In *IEEE symposium on Real Time Systems*, 1989.

[4] Y. Li, W. Wolf. Allocation of Multirate Systems on Multiprocessors with Memory Hierarchy Modeling and Optimization. In *Proceedings, 5th International Workshop on Hardware/Software Co-Design (CODES/CASHE '97)*, 1997.

[5] J. Liedtke, H. Härtig, M. Hohmuth. OS-Controlled Cache PRedictability for Real-Time Systems. In *3rd IEEE Real-Time Technology and Applications Symposium*, 1997.

[6] F. Mueller. Compiler Support for Software-Based Cache Partitioning. In *ACM SIGPLAN Notice*, 1995.

[7] H. Muller, D. Page, J. Irwin, D. May. Caches with Compositional Performance. In *Proceedings, Embedded Processor Design Challenges*, 2002.

[8] G.E. Suh, S. Devadas, L. Rudolph. Analytical Cache Models with Applications to Cache Partitioning. In *Proceedings of the 2001 International Conference on Supercomputing*, 2001.

[9] G.E. Suh, S. Devadas, L. Rudolph. Dynamic Cache Partitioning for Simultaneous Multithreading Systems. In *Thirteenth IASTED International Conference on Parallel and Distributed Computing Systems*, 2001.

[10] J.L. Hennesy, D.A. Patterson. *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 2003.

[11] H.S. Stone, J. Truek, J.L. Wolf. Optimal Partitioning of Cache Memory. In *IEEE Transactions on computers*, volume 41, number 9, 1992.