Memory Hierarchy Studies of Multimedia-enhanced Simultaneous Multithreaded Processors for MPEG-2 Video Decompression

Ulrich Sigmund[1] and Theo Ungerer[2]

[1] VIONA Development GmbH, Karlstr. 27, D-76133 Karlsruhe, uli@viona.de
[2] Institute of Computer Design and Fault Tolerance, University of Karlsruhe, D-76128 Karlsruhe, Germany, ungerer@informatik.uni-karlsruhe.de

This paper explores cache models for a simultaneous multithreaded processor with multimedia enhancements. We start with a wide-issue superscalar processor, enhance it by the simultaneous multithreading (SMT) technique, by multimedia units, and by an additional on-chip RAM storage. Our workload is a multithreaded MPEG-2 video decompression algorithm that extensively uses multimedia units. Various cache models and cache line replacement strategies are simulated to increase performance of the simulated SMT processor models.
Keywords: Simultaneous multithreading, SMT, multimedia extension, MPEG-2 video decompression, cache models

1 Introduction

Today's superscalar processors are able to issue up to six instructions per cycle from a single sequential instruction stream [1]. VLSI technology will soon allow future microprocessors to issue eight or more instructions per cycle. However, ILP found in a conventional instruction stream is limited. Recent studies [2, 3, 4] show the limits of processor utilization even of today's superscalar microprocessors reporting IPC values between 0.14 and 1.9. One solution to increase performance is an additional utilization of more coarse-grained parallelism either by integrating two or more complete processors on a single chip or by using a multithreaded approach. A multithreaded processor is able to pursue multiple threads of control in parallel within the processor pipeline. The functional units are multiplexed between the thread contexts. Most approaches store the thread contexts in different register sets on the processor chip. Latencies are masked by switching to another thread. A simultaneous multithreaded (SMT) processor, in particular, issues instructions from several threads simultaneously. It combines a wide-issue superscalar processor with multithreading. SMT approaches (see e.g. [5-9]) are simulated and evaluated with Spec95 and with database OLTP benchmarks. Most of the simulations show that an 8-threaded SMT reaches a two to threefold IPC increase over single-threaded superscalar processors due to SMT's latency tolerance. In consequence, recent announcements by industry concern a 4-threaded SMT Alpha processor of DEC/Compaq [10] and the MAJC-5200 processor of Sun which features two 4-threaded processors on a single die [11].

Our goal is to evaluate simultaneous multithreading with a multimedia workload. We reported on optimizations for a maximum processor model with an abundance of resources in [12] and on a more realistic processor model in [13]. Other approaches [14, 15] looked at the application of the SMT technique for signal processors, however without applying multimedia instructions. Still compilers do not handle multimedia instructions efficiently. Handcoding is the state-of-the-art of commercially successful video processing algorithms. Such a handcoding was done by our group when transferring a commercial MPEG-2 video decompression algorithm to our SMT multimedia processor model and programming the algorithm in multithreaded fashion. In the following we present a study of cache optimizations for the SMT processor models with a multithreaded MPEG-2 decompression algorithm as workload.

2   The Multithreaded MPEG-2 Video Decompression Algorithm

The MPEG-2 video decompression is partitioned into the following six steps: header decode, Huffman decode, inverse quantization, IDCT (inverse discrete cosine transform), motion compensation, and display. The steps 1 and 2 have to be executed sequentially. The steps 3 to 6 can be executed in parallel for all blocks (and macroblocks) of a single image. Our studies show that the sequential part covers approximately 15.5 % of the executed instructions (depending on the bit rate and size of the encoded material). This results in a theoretical speedup of at most 6.5 compared to the sequential algorithm. The MPEG-2 decompression is made multithreaded by splitting the task into a single parser thread, eight threads for macro block decoding, and an additional display thread. The parser thread executes the first two steps of the decompression and activates up to eight macro block decoding threads that perform steps three to five. The display thread transfers the decompressed images for display into the frame buffer (step 6). The average usage of the instructions assuming a local RAM storage for table look-up (see later) is given in the following table:

Instruction type	Average use (%)
Integer/Multimedia shift and add	54.0
Complex-Integer/Multimedia multiply	3.8
Local load/store	20.1
Global load/store	7.7
Branch	9.9
I/O	2.24
Thread control	1.76

The maximum performance of the decompression is bound by the sequential parser thread. In case of a single local load/store unit, the 20.1% local load/store instructions further restrict the maximum performance to an IPC of approximately five.

3   The Multimedia-enhanced SMT Processor Model

In our approach to combine simultaneous multithreading and multimedia processing we start with a wide-issue superscalar general-purpose processor model based on the 6-stage pipeline of the PowerPC 604, enhance it by simultaneous multithreading, further enhance it by combined integer/multimedia units and by on-chip RAM memory.

The SMT multimedia processor model (see Fig.1) features single or multiple fetch (IF) and decode (ID) units (one per thread), a single rename/issue (RI) unit, multiple, decoupled reservation stations, 10 execution units (four integer/multimedia units, a complex integer/multimedia unit, a branch unit, a thread unit, a global and two local load/store units), a single retirement (RT) and write back (WB) unit, rename registers, a branch target address cache (BTAC), separate I- and D-caches that are shared by all active threads. The four integer/multimedia units share a single reservation station which is able to dispatch four instructions per cycle, all other reservation stations are separate per execution unit. We employ thread-specific instruction buffers (between IF and ID), issue buffers (between ID and RI), and reorder buffers (in front of RT). Each thread executes in a separate architectural register set. However, there is no fixed allocation between threads and (execution) units. The pipeline performs an in-order instruction fetch, decode, rename/issue to reservation stations, out-of-order dispatch from the reservation stations to the execution units, out-of-order execution, and an in-order retirement and write-back.
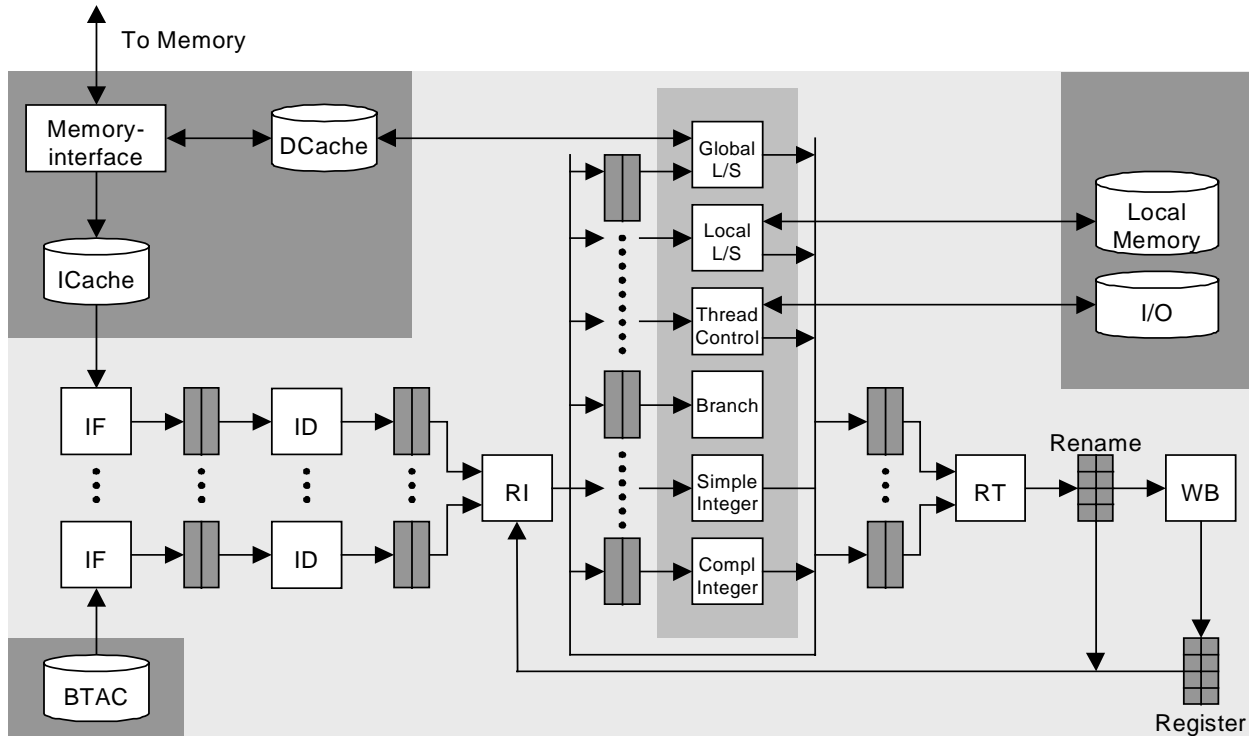
Fig. 1: The SMT Multimedia Processor Model

The rename/issue stage simultaneously selects instructions from all issue buffers up to its maximum issue bandwidth (SMT feature). The integer units are enhanced by MMX-style multimedia processing capabilities (multimedia unit feature). We employ a thread control unit for thread start, stop, synchronization, and for I/O operations. We also employ a 32 KB local on-chip RAM memory (enough for constants and variables of the simulation workload) accessed by two local load/store unit. We use the DLX instruction set enhanced by thread control and by multimedia instructions. We chose to fix the following parameters for all simulations: 32 32-bit general-purpose registers (per thread), 4 MB main memory (enough to store the whole simulation workload), 64-bit system bus, 256 entry reservation stations, 32 entry issue and reorder buffers, D- and I-caches with 32 byte cache lines. In the following we studied the performance impact of different D-cache sizes, D-cache fill burst rates (6:2:2:2 and 32:4:4:4), D-cache preload, D-cache associativity, and D-cache line replacement strategies. We varied the number of threads from 1 to 8 and the issue bandwidth from 1 to 8 in each simulation figure.

The simulator is an execution-based simulator that models all internal structures of the microprocessor model. Two different videos are used as data stream workload for the MPEG-2 algorithm. One to two seconds of video are decompressed by the simulator per simulator run. The produced picture frames can be visually and digitally analyzed to evaluate the correct working of the workload routine and the simulator.

4   Performance Results

4.1   Maximum Processor Models

Software simulation of various configurations of the SMT multimedia processor have been performed to yield a maximum processor model (see [14] for more details). The simulation results of the maximum processor models are shown in Fig. 2. In contrast to the simulations in [14] which are based on static branch prediction, all current simulations apply a McFarling's gshare branch

predictor [16] (with 2 K 2-bit counters and an 8 bit history) which in general yields slightly better results for all models. The misprediction penalty is 5 cycles. In this study we varied the memory hierarchy. In Fig. 2 we present results with a cache fill burst rate of 6:2:2:2 processor cycles. We see that multithreading is very effective if the issue bandwidth is at least four. In particular the two and four-threaded models lead to a high performance increase in the multiple-issue models. There is little improvement for the single-threaded (the contemporary superscalar case) and the two-threaded models when issue bandwidth is increased from 4 to 8.
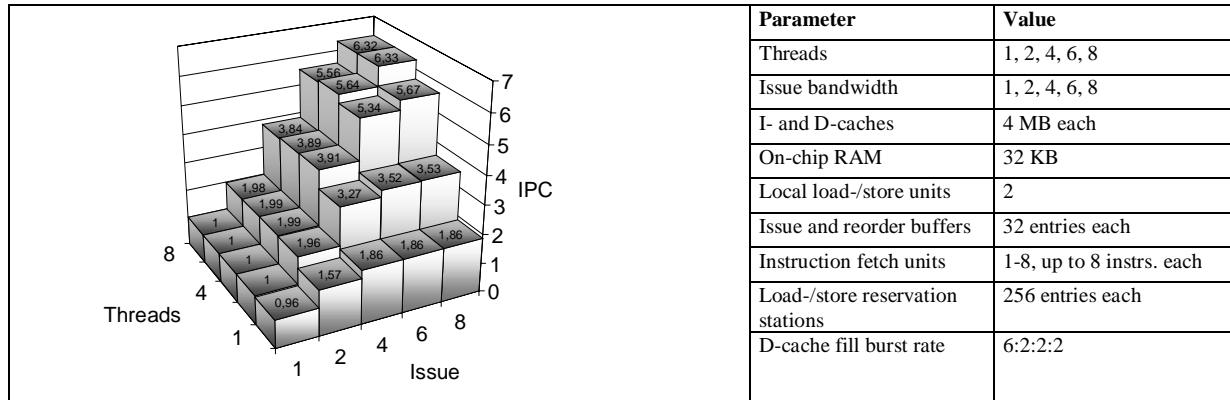

Parameter	Value
Threads	1, 2, 4, 6, 8
Issue bandwidth	1, 2, 4, 6, 8
I- and D-caches	4 MB each
On-chip RAM	32 KB
Local load-/store units	2
Issue and reorder buffers	32 entries each
Instruction fetch units	1-8, up to 8 instrs. each
Load-/store reservation stations	256 entries each
D-cache fill burst rate	6:2:2:2

Fig. 2: Maximum processor models

4.2 Reduced D-Cache

The processor models in Fig. 2 use a 4 MB D-cache (and a 4 MB I-cache) able to host the full workload. The first approach is to reduce the I- and D-cache sizes to realistic values. Because of the small code size of only 24 KB, a reduction of the I-cache size down to 32 or 64 KB yields only an insignificant performance decrease. The reduction of the D-cache size strongly affects overall performance as can be seen in Fig. 3 for the 8-issue models.
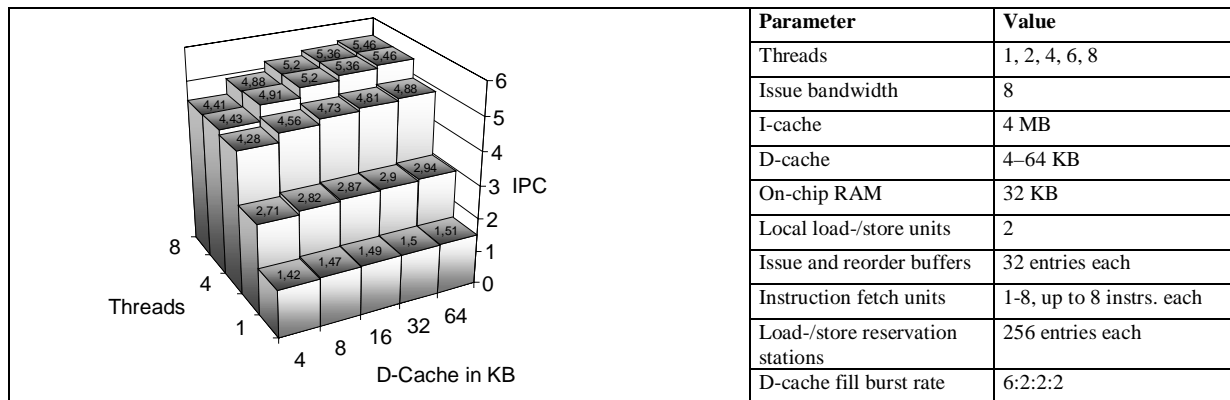

Parameter	Value
Threads	1, 2, 4, 6, 8
Issue bandwidth	8
I-cache	4 MB
D-cache	4–64 KB
On-chip RAM	32 KB
Local load-/store units	2
Issue and reorder buffers	32 entries each
Instruction fetch units	1-8, up to 8 instrs. each
Load-/store reservation stations	256 entries each
D-cache fill burst rate	6:2:2:2

Fig. 3   Reduced D-cache sizes for the 8-issue models

Increasing the data cache size yields better performance. Performance of the 4- and the 8-threaded models exhibit only a small performance increase similar as in Fig. 1.

4.3 Longer Memory Latencies

The pressure on the memory system is increased by simulating longer memory latencies. Instead of a burst rate of 6:2:2:2 processor cycles applied in the simulations in Fig 3, we now apply a burst rate of 32:4:4:4 which is also more realistic for contemporary and near future processors.
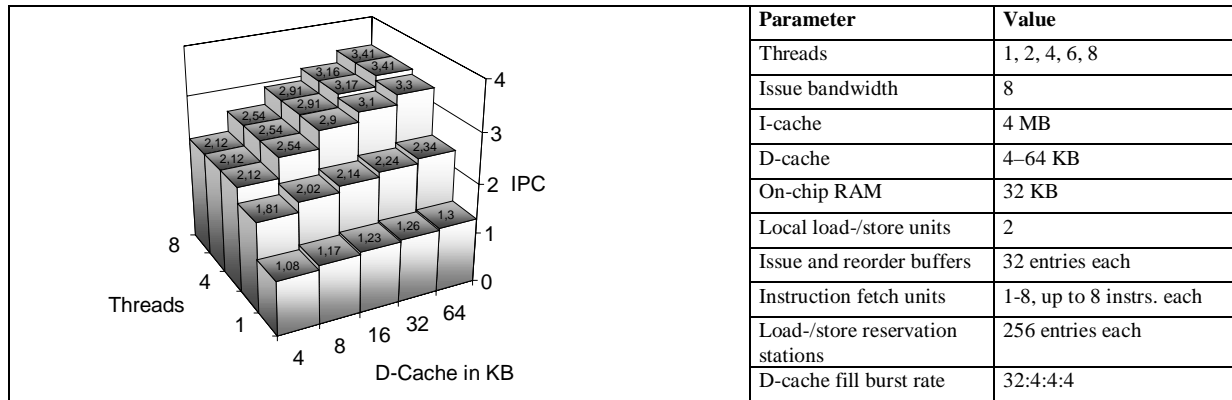
Parameter	Value
Threads	1, 2, 4, 6, 8
Issue bandwidth	8
I-cache	4 MB
D-cache	4–64 KB
On-chip RAM	32 KB
Local load-/store units	2
Issue and reorder buffers	32 entries each
Instruction fetch units	1-8, up to 8 instrs. each
Load-/store reservation stations	256 entries each
D-cache fill burst rate	32:4:4:4

Fig. 4   Reduced D-cache sizes and longer memory latencies

The results in Fig. 4 show a significant performance decrease if compared to Fig. 3. The cache hit ratio of 76% for the 64 KB D-cache for all 8-issue models is relatively low. On the other hand the inspection of the integer/multimedia reservation station shows a high utilization with 69% to 70% for the 8- to 4-threaded models when assuming a 64 KB D-cache (with even higher utilization for smaller D-cache sizes). With 4 integer/multimedia units available, this means that integer/multi-media instructions are blocked within the reservation station because of missing data operands that are not yet loaded from memory.

4.4   Reduced D-cache and Speculative Data Preload

Next we applied the technique of speculative data preload. The motivation behind the preload approach is the regular cache access pattern the can be seen in the cache footprint in Fig. 5 (time scale of memory accesses in horizontal scale, accessed memory addresses in vertical scale). The technique is further motivated by the stream buffer technique [17] which is designed for a stride-based prefetch utilizing long two-dimensional data streams which are commonly used in scientific computing algorithms. Unfortunately, stream buffers cannot be applied in our case. The high initialization overhead of stream buffers is prohibitive for MPEG-2 streams, which are numerous and limited to 16 by 16 bytes only.
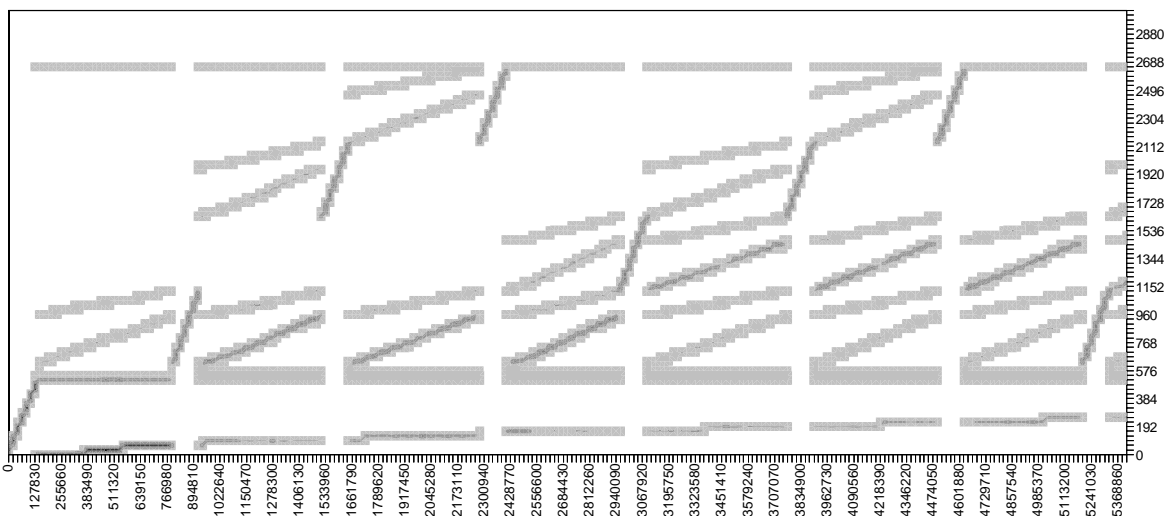

Fig. 5   D-cache footprint

In our approach, the stride and direction of accessed addresses are stored with each D-cache line and an automatic preload of the next cache line is performed. In our implementation each cache line is enhanced by an index storing the last accessed address within the cache line and a speculation state that supports the states "undirected access", "likely forward", "forward", "likely backward", and "backward". Figure 6 shows nearly no performance change, despite the fact that an increase of the cache hit percentage can be observed for the speculative preload models. This is due to the frequent mispredicted preloads, that cause the already busy system data bus to be even more overutilized. This is another indication that speculation can hurt a multithreaded processors performance [13]. Therefore the simulations shown in Figs. 7 to 9 are performed without the preload strategy.
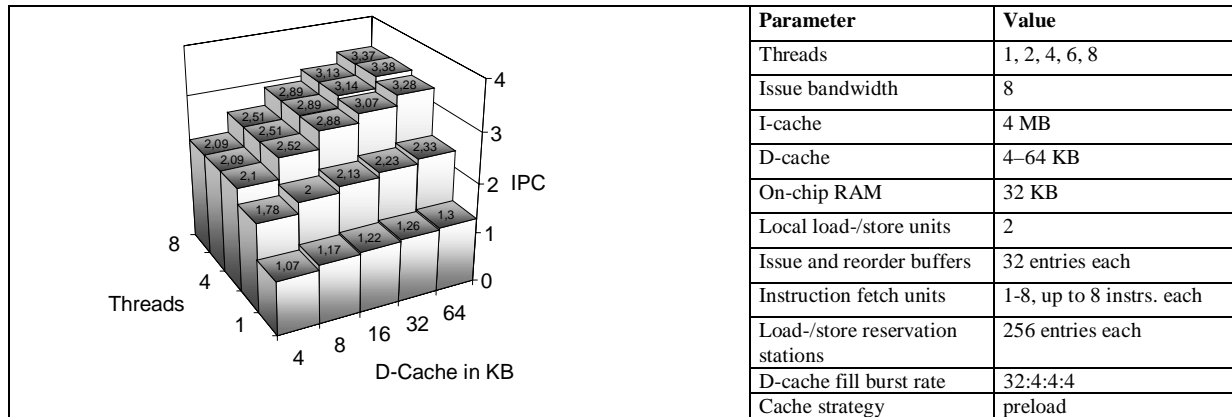

Parameter	Value
Threads	1, 2, 4, 6, 8
Issue bandwidth	8
I-cache	4 MB
D-cache	4–64 KB
On-chip RAM	32 KB
Local load-/store units	2
Issue and reorder buffers	32 entries each
Instruction fetch units	1-8, up to 8 instrs. each
Load-/store reservation stations	256 entries each
D-cache fill burst rate	32:4:4:4
Cache strategy	preload

Fig. 6   Reduced D-cache sizes and speculative preload

4.5   D-cache Associativity

We analyzed the impact of cache associativity next. Up to now we applied 4-way set associative caches. Fig. 7 shows the 2-way set associative D-cache and Fig. 8 the 8-way set associative case.
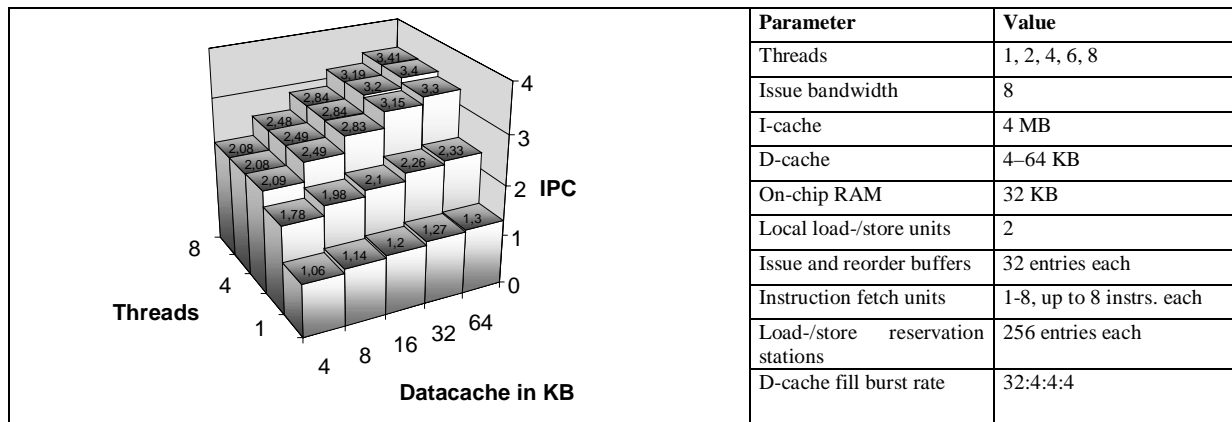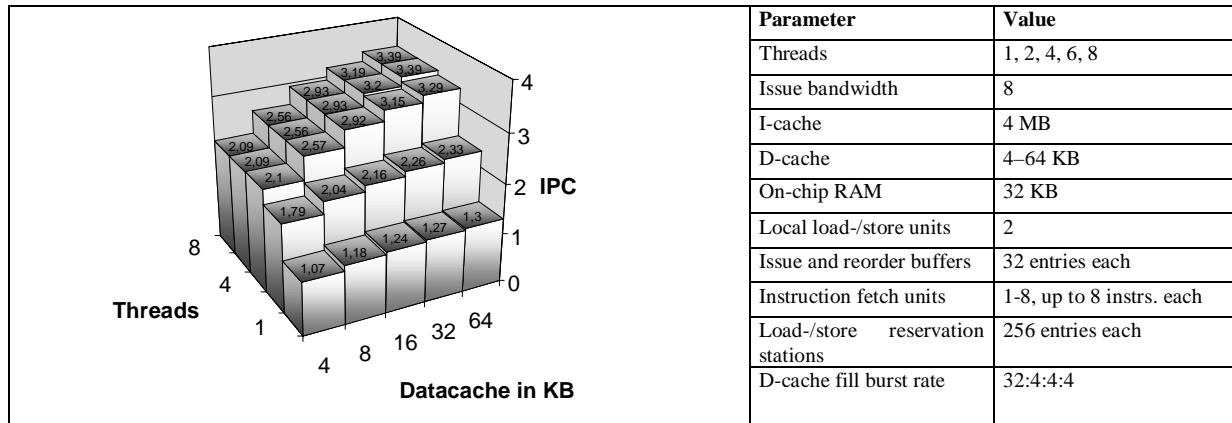

Parameter	Value
Threads	1, 2, 4, 6, 8
Issue bandwidth	8
I-cache	4 MB
D-cache	4–64 KB
On-chip RAM	32 KB
Local load-/store units	2
Issue and reorder buffers	32 entries each
Instruction fetch units	1-8, up to 8 instrs. each
Load-/store reservation stations	256 entries each
D-cache fill burst rate	32:4:4:4

Fig. 7 2-way set associative D-cache

Parameter	Value
Threads	1, 2, 4, 6, 8
Issue bandwidth	8
I-cache	4 MB
D-cache	4–64 KB
On-chip RAM	32 KB
Local load-/store units	2
Issue and reorder buffers	32 entries each
Instruction fetch units	1-8, up to 8 instrs. each
Load-/store reservation stations	256 entries each
D-cache fill burst rate	32:4:4:4

Fig. 8   8-way set associative D-cache

Comparing Figs. 7 and 8, we see a small performance decrease despite the increase of associativity for the 64 KB D-cache sizes and only a small decrease for D-cache sizes with less than 32 KB. Here we assume that the LRU strategy used as replacement strategy for cache lines is not optimal.

4.7   D-cache Line Replacement Strategies

The next investigation concerned the cache line replacement strategy within the D-cache. We designed and simulated several alternative cache line replacement strategies with the aim to prevent replacement of the parser tables from the D-cache (see Fig. 9):

- Strategies 0, 1, and 2: The conventional round-robin, pseudo-random, and LRU strategies.
- Strategy 3: An instruction address based strategy that allows all threads to access the whole D-cache, but restricts access to cache locations for specific instructions. The lower order bits of the instruction address are used for data cache line selection. The idea is to prevent single instructions from covering large data cache areas which may lead to increased thrashing.
- Strategy 4: A thread-ID based strategy aimed to avoid thrashing when different threads access cache-lines that mapped to the same cache location. The lower order bits of the thread ID are used for D-cache line selection.
- Strategy 5 combines strategies 3 and 4.
- Priority: Strategy 6 preferably replaces cache lines of threads with low priority. A cache line of a high priority thread is replaced only if no cache line of a lower priority thread is present in the cache set. We selected the parser thread as high priority thread to avoid dislocation of the parser tables by data of other threads.
- Priority and Random: Strategy 7 implements a less restrictive priority-based strategy. Cache lines of low priority threads are replaced with a probability twice as high as for the high priority thread. A random strategy is used to select cache lines with the same priority for replacement.
- Priority and LRU: Strategy 8 combines Strategy 7 with an extended LRU instead of a random strategy. Cache lines of lower priority threads age twice as fast as cache lines of higher priority threads. Cache lines with the same LRU counter value are selected based on the priority.
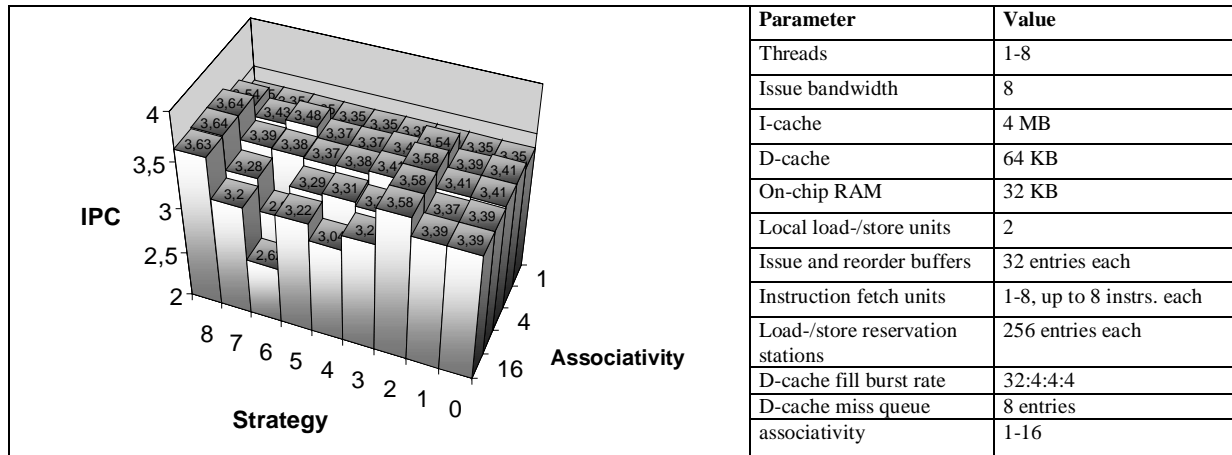
Parameter	Value
Threads	1-8
Issue bandwidth	8
I-cache	4 MB
D-cache	64 KB
On-chip RAM	32 KB
Local load-/store units	2
Issue and reorder buffers	32 entries each
Instruction fetch units	1-8, up to 8 instrs. each
Load-/store reservation stations	256 entries each
D-cache fill burst rate	32:4:4:4
D-cache miss queue	8 entries
associativity	1-16

Fig. 9: IPC of various cache line replacement strategies

Simulations of these strategies (Fig. 9) assuming different associativities for the D-cache show that the importance of the strategy selection rises with the associativity. In particular strategies 4 and 6 show a significant performance degradation, whereas the LRU-based strategies 2 and 8 profit from a higher associativity. The potential disadvantage of the thread priority based strategies 6, 7 and 8 is that the D-cache fills with cache lines of the high priority thread thus limiting cache space for the other threads. The basic thread priority based strategy 6 suffers most. However, strategy 8 which combines a thread priority based strategy with LRU performs best. A look at the cache hit rate shows that the overall hit rate of the best strategy—strategy 8 (Priority and LRU)—is smaller than the hit rate of strategy 2 (LRU). The higher IPC rating of strategy 8 results from the high hit rate of the priority thread—the parser thread—yielding a more balanced parallel execution. A 4-way set-associative D-cache reaches a near optimal result, a further increase of the associativity yields only a small performance increase for the good strategies. Simulations with the lower D-cache fill burst rate of 6-2-2-2 show similar patterns for all strategies, however with less difference in the IPC values of different associativities within each strategy.

5   Conclusions

We simulated various SMT multimedia processor models using a hand-coded multithreaded MPEG-2 video decompression algorithm as workload. Our architectural optimizations targeted D-cache models. Results show a high IPC increase of the SMT models over single-threaded models caused by the latency hiding capability of SMT, but also the sensitiveness of SMT processor performance with respect to memory bandwidth. The simulations with our benchmark program showed that an 8-threaded 8-issue processor model with a 64 KB D-cache size reaches an IPC of 5.46 in case of a 6:2:2:2 D-cache fill burst rate and a IPC of 3.41 in case of 32:4:4:4. The latter IPC value is improved to 3.64 by a combined LRU and priority-based cache line replacement strategy. The cache line replacement strategy becomes more important when memory bandwidth is more restricted. The combined LRU and priority-based replacement strategy that gives highest priority to the parser thread shows the best performance. Our processor models are specifically tailored to the characteristics of our multithreaded MPEG-2 algorithm. Other workloads might favor different configuration, in particular, if the workload consists of unrelated threads of control that do not share any data.

6   References

[1] Silc, J., Robic, B., Ungerer, Th.: Processor Architecture - From Dataflow to Superscalar and Beyond. Springer-Verlag, Berlin, Heidelberg, New York, 1999.

[2] Bhandarkar, D., Ding, J.: Performance Characterization of the Pentium Pro Processor. 3rd Int. Symp. on High-Performance Computer Architecture HPCA-3, San Antonio, Feb. 1997.

[3] Keeton, K., Patterson, D.A., He, Y.Q., Raphael, R.C., Baker, W.E.: Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads. 25th Ann. Int. Symp. on Computer Architecture. Barcelona, Spain, June-July 1998, 15-26.

[4] Barroso, L.A., Gharachorloo, K., Bugnion, E.: Memory System Characterization of Commercial Workloads. 25th Ann. Int. Symp. on Computer Architecture. Barcelona, Spain, June-July 1998, 3-14.

[5] Tullsen, D. M., Eggers, S. J., and Levy, H. M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism. 22nd Ann. Int. Symp. on Computer Architecture. Santa Margherita Ligure, Italy, July 1995, 392-403.

[6] Tullsen, D. M., Eggers, S. J., Levy, H. M., Jo, J. L., and Stamm, R. L.: Exploiting Choice: Instruction Fetch And Issue on an Implementable Simultaneous Multithreading Processor. 23rd Ann. Int. Symp. on Computer Architecture. Philadelphia, May 1996, 191-202.

[7] Sigmund, U., Ungerer, Th.: Evaluating A Multithreaded Superscalar Microprocessor Versus a Multiprocessor Chip. 4th PASA Workshop–Parallel Systems and Algorithms. Forschungszentrum Jülich, Germany, World Scientific Publishing, April 1996, 147-159.

[8] Gulati, M., Bagherzadeh, N.: Performance Study of a Multithreaded Superscalar Microprocessor. 2nd Int. Symp. on High-Performance Computer Architectures, February 1996, 291-301.

[9] Lo, J. L., Barroso, L. A., Eggers, S. J., Gharachorloo, K., Levy, H. M., and Parekt, S. S.: An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors. 25th Ann. Int. Symp. on Computer Architecture. Barcelona, Spain, June-July 1998, 39-50.

[10] Emer, J.: Simultaneous Multithreading: Multiplying Alpha's Performance. Microprocessor Forum 1999, San Jose, Ca., Oct. 1999.

[11] Gwennap, L.: MAJC Gives VLIW a New Twist. Microprocessor Report, Sept. 13, 1999.

[12] Oehring, H., Sigmund, U., Ungerer, Th.: Simultaneous Multithreading and Multimedia. Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC99, in connection with the HPCA-5 Conf., Orlando, Jan. 9, 1999.

[13] Oehring, H., Sigmund, U., Ungerer, Th.: MPEG-2 Video Decompression on Simultaneous Multithreaded Multimedia Processors. 1999 Int. Conf. on Parallel Architectures and Compilation Techniques (PACT '99), Newport Beach, Ca., Oct. 1999, 11-16.

[14] Wittenburg, J.P., Meyer, G., Pirsch, P.: Adapting and extending Simultaneous Multithreading for High Performance Video Signal Processing Applications. Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC99, in connection with the HPCA-5 Conf., Orlando, Jan. 9, 1999.

[15] Pontius, M., Bagherzadeh, N.: Multithreaded Extensions Enhance Multimedia Performance. Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC99, in connection with the HPCA-5 Conf., Orlando, Jan. 9, 1999.

[16] McFarling, S.: Combining Branch Predictors. DEC WRL Technical Note TN-36, DEC Western Research Laboratory 1993.

[17] Palarcharla, S., Kessler, R.E.: Evaluating Stream Buffers As a Secondary Cache Replacement. 21st Int. Symp. on Computer Architecture. April 1994, 24-33.