

# Simultaneous Multithreading and Multimedia

Heiko Oehring<sup>1</sup>, Ulrich Sigmund<sup>1</sup>, and Theo Ungerer<sup>2</sup>

<sup>1</sup> VIONA Development GmbH, Karlstr. 27, D-76133 Karlsruhe, {heiko,uli}@viona.de

<sup>2</sup> Institute of Computer Design and Fault Tolerance, University of Karlsruhe,  
D-76128 Karlsruhe, ungerer@informatik.uni-karlsruhe.de

This paper explores microarchitecture models for a simultaneous multithreaded processor with multimedia enhancements. We start with a wide-issue superscalar processor, enhance it by the simultaneous multithreading 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. The simulations showed that a single-threaded, 8-issue maximum processor (assuming an abundance of resources) reaches an IPC (instructions per cycle) count of only 1.60, while an 8-threaded 8-issue processor is able to reach an IPC of 6.07. A more realistic processor model reaches an IPC of 1.27 in the single-threaded 8-issue vs. 3.21 in the 8-threaded 8-issue model. So, the generalizing conclusion is that an 8-threaded 8-issue processor may yield up to threefold performance increase over the single-threaded 8-issue model.

Keywords: Simultaneous multithreading, multimedia extension, MPEG-2 video decompression

## 1 Introduction

Current microprocessors utilize instruction-level parallelism (ILP) by a deep processor pipeline and by the superscalar instruction issue technique [1]. A superscalar processor is able to issue multiple instructions each clock cycle from a conventional linear instruction stream. DEC Alpha 21164, IBM PowerPC 604, MIPS R12000, Sun UltraSPARC-II, and HP PA-8500 issue up to four instructions per cycle from a single thread. DEC Alpha 21264 is already able to issue six instructions per cycle. VLSI-technology will soon allow future microprocessors to issue 8 or more instructions per cycle.

As the issue rate of future microprocessors increases, the compiler or the hardware will have to extract more ILP by analyzing a larger instruction window. However, ILP found in a conventional instruction stream is limited. In general, integer-dominated programs feature a rather low ILP, while a high ILP can be extracted from floating-point programs by transforming loop-level parallelism into ILP using compiler techniques like loop unrolling or software pipelining.

Recent studies showed the limits of processor utilization even of today's superscalar microprocessors. CPI (cycles per instruction) values between 0.5 and 1.5 have been reported by Bhandarkar and Ding [2] for SPEC95 benchmark programs on the PentiumPro. Measurements by Keeton et al. [3] of commercial online transaction processing (OLTP) database workloads on a quad PentiumPro symmetric multiprocessor show an overall CPI of 3.39. Similar measurements by Barroso et al. [4] show an even less favorable CPI of 7.0 (IPC of only 0.14) for an OLTP and a CPI of 1.3 till 1.9 for a decision support system (DSS) workload on a four-issue Alpha 21164 processor. The low IPC stems from the 75% of time that is spent stalling for memory accesses due to the large working set common in database transaction processing that leads to a high cache miss ratio even of a large secondary-level cache.

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, in general, stores multiple thread contexts in different register sets on the processor chip and multiplexes the functional units between the contexts. Context switching is very fast and thereby all kinds of latencies can be bridged by switching to another thread. A simultaneous multithreaded (SMT) processor, in particular, can issue instructions from several threads simultaneously. It combines a wide-issue superscalar processor with multithreading. SMT approaches (see e.g. [5-10]) are simulated and usually evaluated with Spec92 and Spec95 benchmarks, and recently with database on-line transaction processing (OLTP) and decision support (DSS) workloads [10]. Most of the simulations show that SMT reaches an up to threefold IPC increase over single-threaded superscalars due to SMT's latency tolerance.

However, simultaneous multithreading is not yet evaluated with a multimedia workload! Still compilers do not handle multimedia instructions efficiently. Therefore performance results of a compiled jpeg benchmark of SPECint95 or of compiled programs stemming from the MediaBench benchmark suite [11] are not representative for high performance real-time media processing applications.

## 2 Media Processing

Media processing (digital multimedia information processing) is the decoding, encoding, interpretation, enhancement, and rendering of digital multimedia information. Today's video and 3D graphics require high bandwidth and processing performance. Multimedia units employ SIMD instructions, saturation arithmetic, and additional arithmetic, masking, selection, reordering and conversion instructions.

The MPEG-2 video algorithm is defined in ISO/IEC 13818-2 [12]. MPEG-2 is the video compression standard that has been chosen for digital TV (cable, satellite, terrestrial broadcast), DVD and HDTV. It provides high quality video with data rates of 2 – 20 Mbits/s. The MPEG-2 video decompression is a good example for a class of video stream algorithms but not for 3D applications. It combines high computational parts like the IDCT (inverse discrete cosine transformation), completely memory bound elements like the motion compensation with highly irregular program flows in the Huffman decoder. This mix is representative for many multimedia compression and decompression algorithms. We have chosen the MPEG-2 video decompression as workload, because it is a typical real world application, and provides enough parallelism to fully utilize a SMT processor.

The MPEG-2 video decompression can be divided into the following six steps:

1. Header decode provides video sequence parameters such as picture rate, bit rate, image size, structure and decoding parameters.
2. Huffman decode decodes variable-length codes into fixed length numbers, which represent quantized IDCT coefficients, scaling factors, and motion vectors. The step includes run-length decoding of zeros for the DCT coefficients.
3. Inverse quantization multiplies coefficients by quantizer factors to restore them to the original range.

4. IDCT changes each 8x8 block of IDCT coefficients to convert the data from the frequency domain back to the original spatial domain. This gives the actual pixel values for I-blocks, but only the differences for each pixel for P- and B-blocks.
5. Motion compensation adds the differences in the IDCT step to the pixels in the reference block as determined by the motion vector for P-blocks, and to the average of the forward and backward reference blocks for B-blocks.
6. Display converts color from YCbCr coordinates to RGB color coordinates, including upsampling Cb and Cr values, and writing to the frame buffer for displaying the decoded video.

The steps 3 to 6 can be executed in parallel for all blocks (and macroblocks) of a single image. The steps 1 and 2 have to be executed sequentially. Our studies show that the non parallel part covers approximately 14 to 17% of the executed instructions (depending on the bit rate and size of the encoded material). This results in a theoretical speedup of 6 to 7 compared to a sequential algorithm.

Our MPEG-2 decompression algorithm is made multithreaded and fully hand coded in assembly language. It is based on a commercially available MPEG 2 decoder for INTEL PCs. The MPEG-2 decoding algorithm does highly benefit from special multimedia instructions.

By applying simultaneous multithreading to multimedia streams we envision better utilization of the multimedia units, because the multimedia instructions can only be used in specific stages of a single-threaded MPEG-2 algorithm.

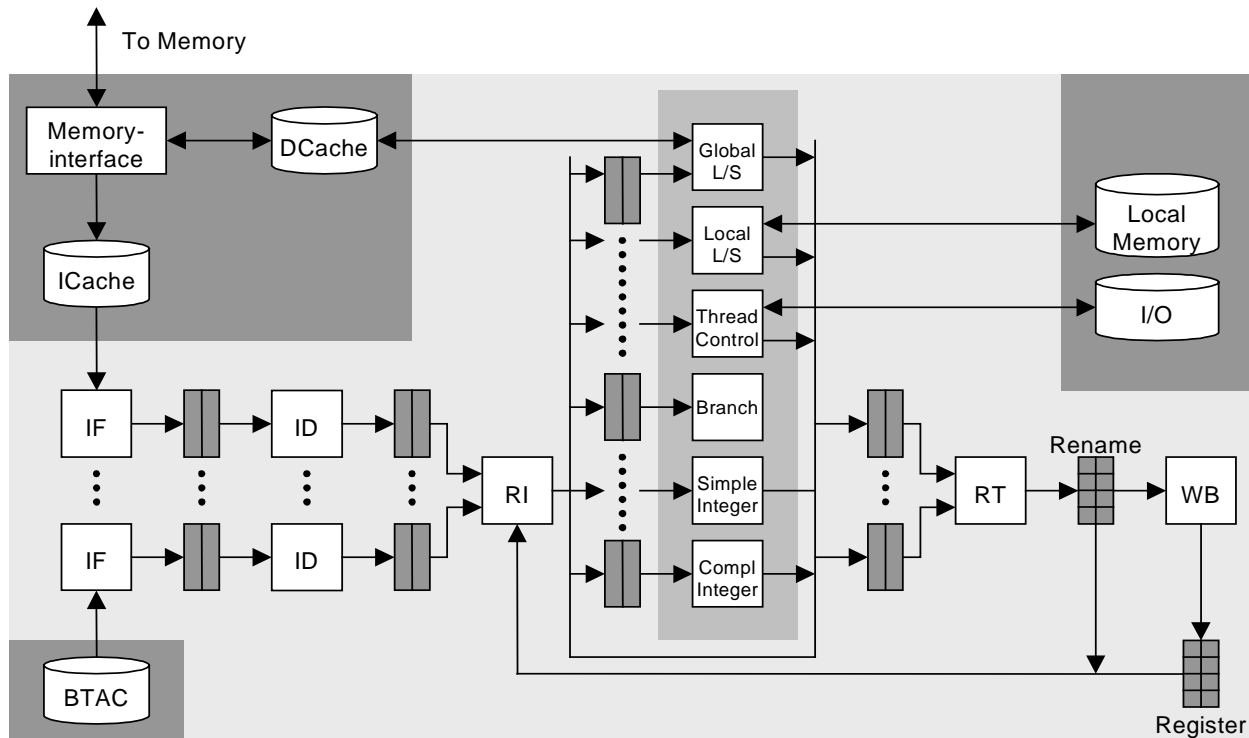
## **3 Experimental Setup**

### **3.1 Overview of the SMT Multimedia Processor Model**

In our approach to combine simultaneous multithreading and multimedia we start with a wide-issue superscalar general-purpose processor model based on 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, a single rename/issue (RI) unit, multiple, decoupled reservation stations, multiple execution units, in particular, up to four combined integer/multimedia units, a complex integer/multimedia unit, a branch unit, separate local and global 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. 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.

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 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 local RAM memory accessed by the local load/store unit.



**Fig. 1: The SMT Multimedia Processor Model**

Simplifications concern the instruction set. We use the DLX instruction set enhanced by thread control and by multimedia instructions. No floating-point unit and static instead of dynamic branch prediction is applied.

A floating-point unit is not necessary for MPEG-2, but would be necessary for 3D applications. Previous work showed that complex dynamic branch prediction and speculation techniques are effective for SMT processors, but by far not as effective as for single-threaded multiple-issue processors, because the SMT feature bridges branch latencies by multithreading. The branches in the MPEG-2 decompression algorithm are well suited for a static prediction (either with a high bias, or completely random), so we do not expect a significant performance degradation when simulating with static instead of dynamic branch prediction.

### 3.2 The Simulator

The simulator is an execution-based simulator that models all internal structures of the microprocessor model. Software simulation of various configurations of the SMT multimedia processor are performed. Since we cannot vary all parameters, 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, 4-way set-associative D- and I-caches with 32 byte cache lines and a cache fill burst rate of 6-2-2-2 processor cycles, and 32 KB local on-chip RAM (enough for constants and variables of the simulation workload).

We primarily vary the number of threads from 1 to 8 and the issue bandwidth from 1 to 8. We further assume different values for the number and size of issue buffers, reservation stations, reorder buffers, size of BTAC, the number of integer/multimedia units, number of result buses and rename registers, the size and refill strategies of the D-cache.

### 3.3 The Workload

Multimedia applications are usually partly compiler-generated and partly hand-coded in assembly language. The hand-coded part concerns the use of the multimedia instructions and some time critical routines. Our workload is a multithreaded version of the MPEG-2 video decompression algorithm. The task is split between a single parser thread, that calls up to eight threads for macro block decoding. An additional display thread is used to transfer the decoded images for display into the frame buffer. The maximum performance of the decoder is bound by the sequential parser thread. The applied instruction mix 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
Thread control (including I/O)	4.5

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

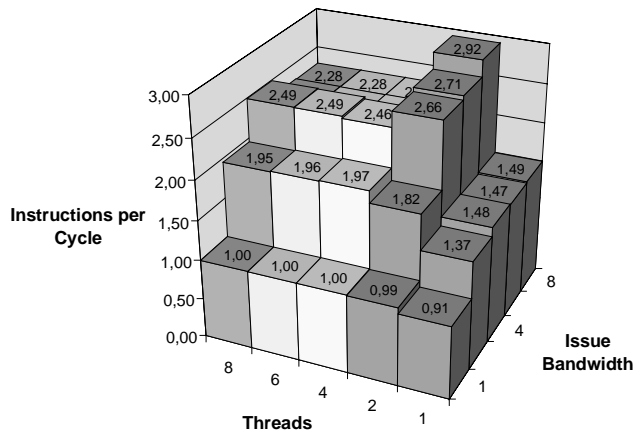
Our evaluation proceeded in two steps. First, we developed and optimized a maximum processor that includes as much resources as a potential processor in 10 years. Second, we transferred the experimental results to a realistic processor with a resource capacity of a next generation processor.

### 4.1 Maximum Processor

Our initial maximum processor featured a 1024-entry BTAC, 1024 rename registers, a 32-entry issue buffer per thread, 4 simple integer/multimedia units, 256-entry reservation stations separate for each execution unit, an own result bus per execution unit, and 256-entry reorder buffers. Performance was a disappointing IPC of 2.28 for the 8-threaded 8-issue model, with slightly better results for the 6-issue and for the 2-threaded models (see Fig. 2).

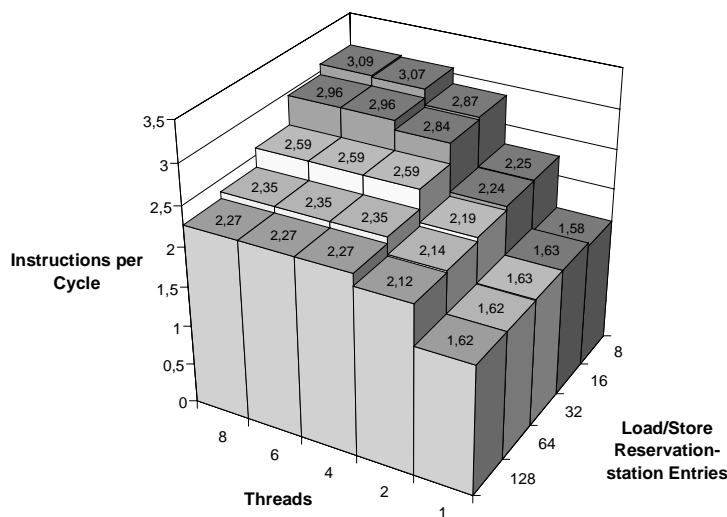
We found out that smaller sizes of the reservation stations, in particular of the reservation stations for the thread unit, the global and the local load/store units (assuming now 16 entries each), increased the IPC to 2.96 in the 8-threaded 8-issue case (see Fig. 3). This was a surprise, because the usual assumption would suggest a better performance with larger buffers. However, we assume large reservation stations (and large reorder buffers) draw too many highly speculative instructions into the pipeline. Smaller buffers limit the speculation depth of fetched instructions and lead to the fact that only non-speculative instructions or instructions with low speculation depth are fetched, decoded, issued and executed in a SMT processor. An abundance of speculative instructions does harm to the performance of a SMT processor. Another reason for the negative effect of large reservation stations for the load/store and thread control units lies in the fact, that those instructions have a long latency, and typically have two to three integer

instructions as consumer. The effect is, that the consuming integer instructions eat up space in the integer reservation station, thus blocking instructions from other threads from entering it. This multiplication effect is made even worse by the non speculative execution of store and thread control instructions.



Parameter	Value
Threads	1-8
Issue Bandwidth	1-8
Caches	4 MB each
Local Memory	32 KB
Instruction Fetch Units	1-8, up to 8 instrs. each
Reservation Stations	256 entries each
Functional Units	9
Result Buses	8

**Fig. 2: IPC of the maximum processor: initial model**



Parameter	Value
Threads	1-8
Issue Bandwidth	8
Load-/Store Reservation Stations	8-128 entries each
Simple Integer Reservation Stations	256 entries each

**Fig. 3: IPC of the maximum processor: smaller reservation stations**

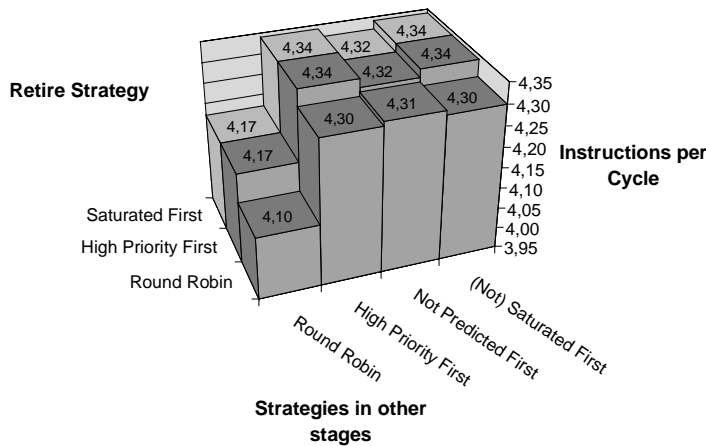
Next we optimized the integer/multimedia reservation stations. Each of the four integer/multimedia units featured a separate reservation station with 256 entries. Instruction were preferably issued to the reservation station of the same integer unit, leading to a overflow of the corresponding reservation station. We unified the reservation station to a single, common, 256-entry reservation station for all four integer units able to provide four instructions per cycle to the different integer units. This architectural change provided a performance increase from 2.96 to 3.27 in the 8-issue 8-threaded model (even more for other models).

Up to now a non-blocking cache model is provided; loads and stores are performed out of order unless an address conflict arises. However, load or store operations never pass another memory operation of which the address is not yet computed. So the instructions in the reservation

stations of the local and the global load/store unit are blocked even for loads and stores of other threads. The next optimization is to relax access ordering so that loads and stores of other threads can pass loads or stores with unavailable memory addresses, while succeeding loads and stores of the same thread are blocked. This showed an improvement of 0.83 to an IPC of 4.1 for the 8-threaded 8-issue case.

Our next survey regarded different thread selection strategies. Tullsen et al. [6] examined instruction fetch policies that give highest fetch priority to the threads that are least likely to be on a wrong path (BRCOUNT), have the fewest outstanding data cache misses (MISSCOUNT), have the fewest instructions in the decode, rename and queue pipeline stages (ICOUNT), and a policy that gives lowest priority to threads that have the oldest instructions in integer and floating-point queues (IQPOSN). The combining strategies ICOUNT and IQPOSN are better than the strategies that evaluate only a single cause (BRCOUNT and MISSCOUNT), and all are better than the basic round-robin policy. Issue selection policies like branch-first or speculative-instructions-last showed no significant improvement over the basic oldest-instruction-first policy.

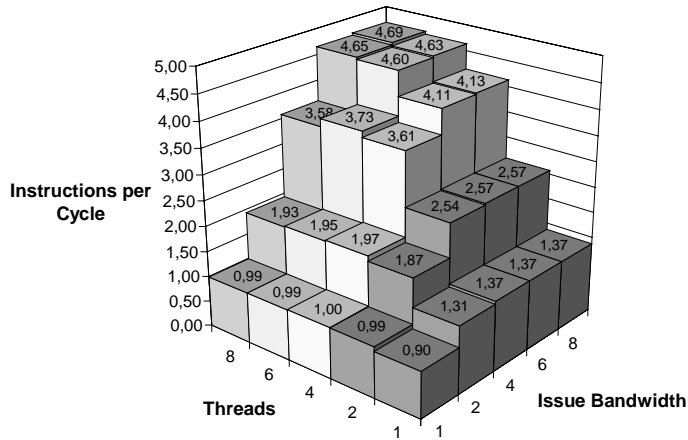
We modified Tullsen et al.'s experiments by controlling the issue, the dispatch from reservation stations, and the retirement due to the three different parameters thread priority, speculation depth, and saturation of the (per thread) issue and reorder buffers. We extracted three strategies: highest-priority-thread-first, non-speculative-instructions-first (similar to BRCOUNT), and non-saturated-first (similar to ICOUNT). Highest priority is given to the parser thread. All three strategies were subsequently applied to the issue, the dispatch from reservation stations, and the retirement stages. We applied as many fetch units as threads in our simulations. All three strategies improved IPC of the 8-threaded 8-issue model by about 0.24 over the basic round robin strategy leading to an IPC of 4.34 for the 8-threaded 8-issue case (see Fig. 4).



Parameter	Value
Threads	8
Issue Bandwidth	8
Caches	4 MB each
Local Memory	32 KB
Instruction Fetch Units	1-8, up to 8 instrs. each
Load-/Store Reservation Stations	16 entries each
Simple Integer Reservation Stations	256 entries each
Functional Units	10
Result Buses	8

**Fig. 4: IPC of the maximum processor with varying issue, dispatch, and retire strategies**

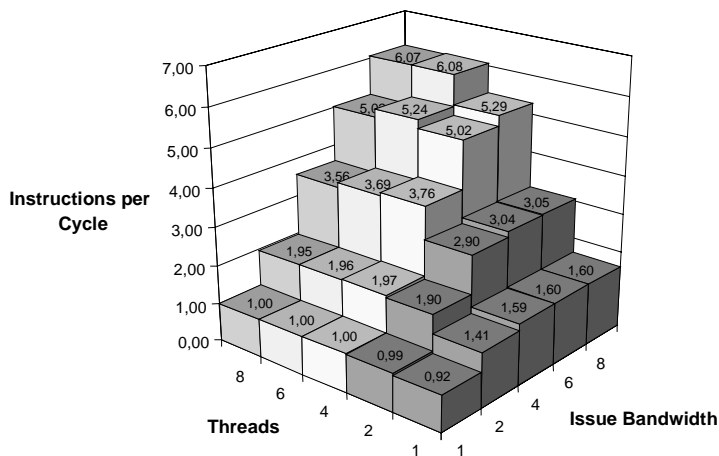
Next we restricted the length of the reorder buffers. Here lengths of 16 or 32 showed as superior to sizes of 8, 64, and much better than 256. The results for 16 entries per thread are shown in Fig. 5. Again, we assume that reorder buffers which are too large draw too many speculative instructions overfilling the issue buffers and the reservation stations with speculative instructions.



Parameter	Value
Threads	8
Issue Bandwidth	8
Caches	4 MB each
Local Memory	32 KB
Instruction Fetch Units	1-8, up to 8 instrs. each
Load-/Store Reservation Stations	16 entries each
Simple Integer Reservation Stations	256 entries each
Functional Units	10
Result Buses	8
Reorder Buffer	16 entries per thread

**Fig. 5: IPC of the maximum processor with 16 entries per thread in the reorder buffer**

Now the main bottleneck is the single local load/store unit which restricts the IPC to at most 5 (there are 20.1% local load/store operations). Here we assume a second local load/store unit and we reach the maximum performance shown in Fig. 6. We see in Fig. 6 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 superscalar case) and the two-threaded models when issue bandwidth is increased from 4 to 8.



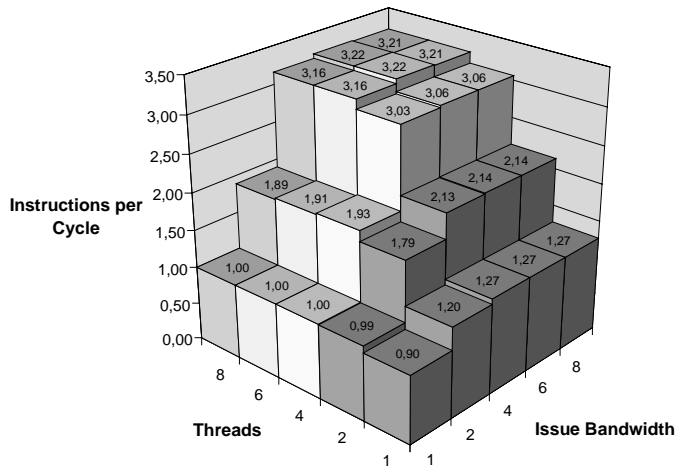
Parameter	Value
Threads	1-8
Issue Bandwidth	1-8
Caches	4 MB each
Local Memory	32 KB
Instruction Fetch Units	1-8, up to 8 instrs. each
Load-/Store Reservation Stations	16 entries each
Simple Integer Reservation Stations	256 entries each
Functional Units	10
Result Buses	8

**Fig. 6: IPC of the maximum processor with on-chip RAM and two local load/store units**

## 4.2 Realistic Processor

Starting with the experiences of the maximum processor we configured a realistic processor as can be implemented in next generation of microprocessors. Several optimization strategies were pursued that cannot be described here in detail. At last we chose the following characteristics: three integer/multimedia units, only three result buses, 128 rename registers, 32 KB primary I- and D-caches, 128-entry BTAC, 16-entry issue buffer per thread, 32 KB on-chip RAM, one local and one global load/store unit, 16-entry reservation stations, a common reservation station for the

simple integer units and another one for the branch unit and the complex integer unit. The simulation results are reported in Fig. 7.



Parameter	Value
Threads	1-8
Issue Bandwidth	1-8
Caches	32 MB each
Local Memory	32 KB
Instruction Fetch Units	1-8, up to 8 instrs. each
Reservation Stations	16 entries for each type
Functional Units	8
Result Buses	3

**Fig. 7: IPC of the realistic processor with on-chip RAM**

## 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 a maximum and a realistic processor model.

The simulations showed that a single-threaded, 8-issue maximum processor (assuming an abundance of resources) reaches an IPC count of only 1.60, while an 8-threaded 8-issue processor reaches an IPC of 6.07. A more realistic processor model reaches an IPC of 1.27 in the single-threaded 8-issue and 3.21 in the 8-threaded 8-issue model. So, the generalizing conclusion is that an 8-threaded 8-issue processor may yield up to threefold performance increase over the single-threaded 8-issue model for a multithreaded MPEG-2 decompression algorithm.

Increasing the issue bandwidth from 4 to 8 yields only a marginal gain (except for the 4-, to 8-threaded maximum processor). Increasing the number of threads from single to 2- or 4-threaded yields a high gain for the 2- to 8-issue model, and a significant gain for the 8-threaded model. 32 KB code and 32 KB data caches are enough, complex cache strategies yield only a marginal gain. On-chip RAM combined with a local load/store unit is effective for all processor configurations and even a second local load/store unit is advantageous.

## 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. Proc. of the 3<sup>rd</sup> International Symposium on High-Performance Computer Architecture HPCA-3, San Antonio, February 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. Proc. of the 25<sup>th</sup> Annual International Symposium on Computer Architecture. Barcelona, Spain, June-July 1998, 15-26.
- [4] Barroso, L.A., Gharachorloo, K., Bugnion, E.: Memory System Characterization of Commercial Workloads. Proc. of the 25<sup>th</sup> Annual International Symposium 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. Proc. of the 22<sup>nd</sup> Annual International Symposium on Computer Architecture. Santa Margherita Ligure, Italy, July 22-24, 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. Proc. of the 23<sup>rd</sup> Annual International Symposium On Computer Architecture. Philadelphia, May 1996, 191-202.
- [7] Sigmund, U., Ungerer, Th.: Evaluating A Multithreaded Superscalar Microprocessor Versus a Multiprocessor Chip. Proc. of the 4<sup>th</sup> PASA Workshop—Parallel Systems and Algorithms. Forschungszentrum Jülich, Germany, World Scientific Publishing, April 10-12, 1996, 147-159.
- [8] Gulati, M., Bagherzadeh, N.: Performance Study of a Multithreaded Superscalar Microprocessor. Proc. of the 2<sup>nd</sup> International Symposium on High-Performance Computer Architectures, February 1996, 291-301.
- [9] Hily, S., Sez nec, A.: Branch Prediction and Simultaneous Multithreading. Proc. of the Parallel Architectures and Compilation Techniques PACT'96, Boston, Oct. 20-23, 1996, 169-173.
- [10] 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. Proc. of the 25<sup>th</sup> Annual International Symposium On Computer Architecture. Barcelona, Spain, June 27 – July 1, 1998, 39-50.
- [11] Lee, C., Potkonjak, M., and Mangione-Smith, W. H.: MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. MICRO-30, Research Triangle Park, Dec. 1-3, 1997, 330-335.
- [12] ISO/IEC 13818-2:1996(E) Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Video. International Standard Organization 1996.