

MPEG-2 Video Decompression on Simultaneous Multithreaded Multimedia Processors

Heiko Oehring
VIONA Development GmbH
Karlstr. 27
D-76133 Karlsruhe, Germany
heiko@viona.de

Ulrich Sigmund
VIONA Development GmbH
Karlstr. 27
D-76133 Karlsruhe, Germany
uli@viona.de

Theo Ungerer
Institute of Computer Design
and Fault Tolerance
University of Karlsruhe
D-76128 Karlsruhe, Germany
ungerer@informatik.uni-
karlsruhe.de

Abstract

This paper explores microarchitecture models for a simultaneous multithreaded processor with multimedia enhancements. We enhance a wide-issue superscalar processor 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. Our simulation results suggest that a 2- or 4-threaded 4-issue processor with a small on-chip RAM accessed by a local load/store unit will be superior to a wide-issue (single-threaded) superscalar processor.

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 contemporary superscalar processor is able to issue up to six instructions each clock cycle from a conventional linear instruction stream. However, ILP found in a conventional instruction stream is limited. One solution to increase performance is an additional utilization of thread-level parallelism by using a simultaneous multithreaded (SMT) processor which is able to issue instructions from several threads simultaneously. Simulations of SMT processors show a two- to threefold IPC increase over single-threaded superscalars for Spec benchmarks (see e.g. [2]) and for database on-line transaction processing and decision support workloads [3] due to SMT's latency tolerance.

Until recently simultaneous multithreading was not yet evaluated with a multimedia workload. Wittenburg et al. [4] looked at the application of the SMT technique for signal processors using combining instructions that are applied to registers of several register sets simultaneously instead of multimedia operations. Simulations

with a Hough transformation as workload showed a speed-up of up to six compared to a single-threaded processor without multimedia extensions. Pontius and Bagherzadeh [5] evaluated a multithreaded superscalar signal processor model using several video decode, picture processing, and signal filter programs as workloads. The programs have been parallelized at source code level by partitioning the main loop and distributing the loop iteration to several threads. The rather disappointing speedup that was reported for multithreading results from algorithmic restrictions and from the already high IPC in the single-threaded model. The latter is only possible because multimedia instructions were not used. Otherwise a large part of the IPC in the single-threaded model would be hidden by the SIMD parallelism within the multimedia instructions.

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 [6] are not representative for high performance real-time media processing applications. Handcoding is still state-of-the-art of commercially successful video processing algorithms.

2 The Multithreaded MPEG-2 Video Decompression Algorithm

The MPEG-2 video decompression 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.

The MPEG-2 video decompression is partitioned into the following six steps: header decode, Huffman decode, inverse quantization, IDCT, 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.

Our MPEG-2 decompression algorithm is based on a commercially available MPEG 2 decoder for INTEL PCs. It is fully hand coded in assembly language and does highly benefit from special multimedia instructions. 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. Cooperative multithreading is applied. The parser thread executes the first two steps of the decoding and activates up to eight macro block decoding threads that perform steps three to five. The display thread transfers the decoded images for display into the frame buffer (step 6).

The average instruction usage assuming on-chip RAM for table look-up is shown in the following table.

The maximum performance of the decoder 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. The parallelization overhead is about 1.76%, which are included in the 4.5% thread control instructions shown in the last row of the 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

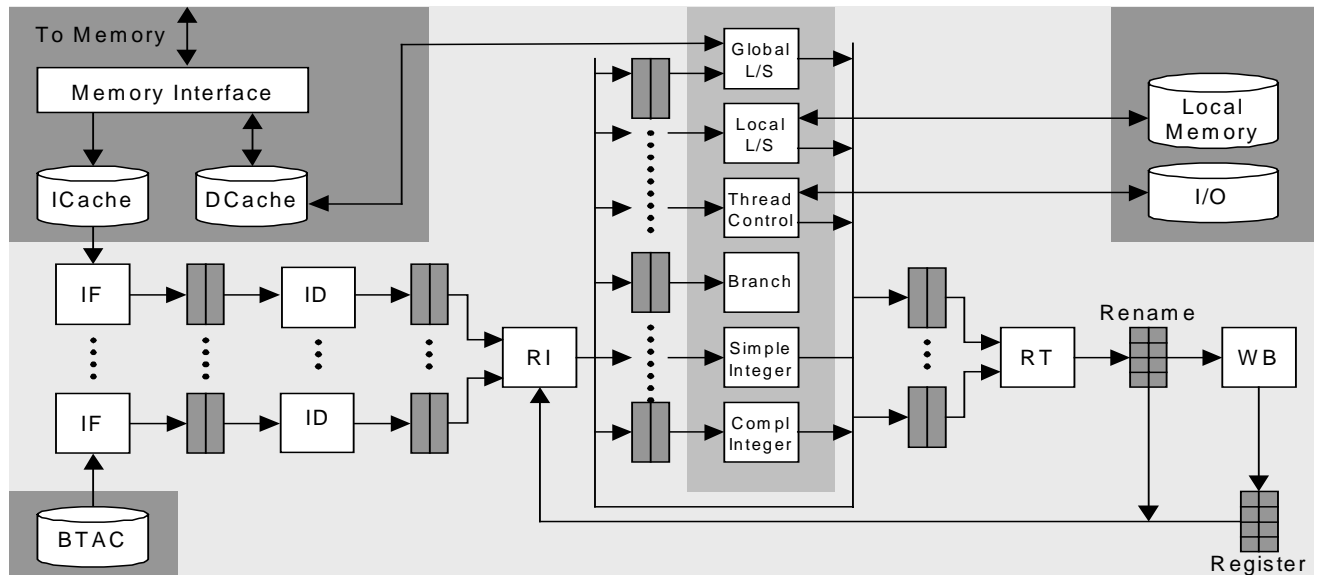


Fig. 1: The SMT Multimedia Processor Model

3 The SMT Multimedia Processor Model

We start with a wide-issue superscalar 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, 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.

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 applying a simple round-robin strategy (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 local RAM memory accessed by the local load/store unit. 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.

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.

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 varied the number of threads from 1 to 8 and the issue bandwidth from 1 to 8. We further assumed 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.

4 Performance Results

Our evaluation proceeded in two steps. First, we developed and optimized a maximum processor that included an abundance of resources (see [7]). Next we configured a realistic processor as it can be implemented in next generation of microprocessors. The simulation results of the optimized maximum processor are shown in Fig. 2. 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. The most surprising finding was that smaller reservation stations for the thread unit, the global and the local load/store units as well as smaller reorder buffers increased the IPC value for the multithreaded models (see al. [7] for more details). In the rest of the paper we focus on the development of a more realistic processor with a resource capacity of a next generation processor.

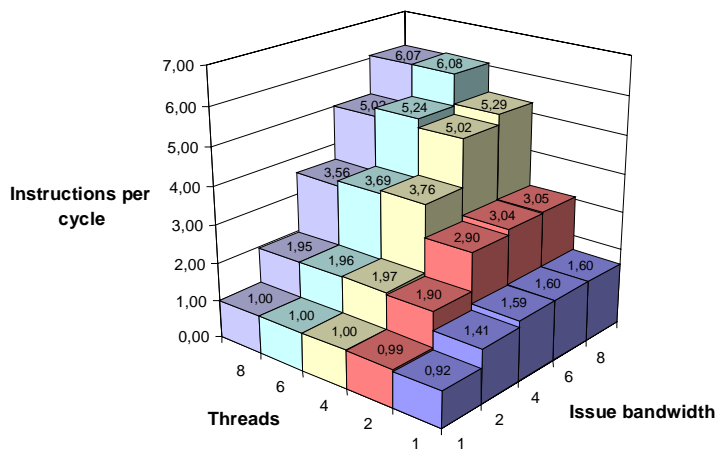


Fig. 2: IPC of the maximum processor with on-chip RAM and two local load/store units

The following design decisions were made:

- A look at the instruction mix shows that three simple integer/multimedia units suffice for the 54% integer and multimedia instructions.
- The number of result buses can be limited to three, because several units, as e.g. the complex integer/multimedia or the thread control unit, are rarely used (IPC increases by 0.01 in the 8-threaded 8-issue up to 0.14 in the 4-threaded 4-issue model with separate result buses).

- The set of physical registers is only utilized by at most 10%. Thus we assume 128 instead of 1024 registers.
- A smaller cache size does harm the performance. I- and D-caches of 32 KB each are realistic.
- The BTAC is reduced from 1024 to 128 entries.
- The number of reservation stations is limited to five: one for the local load/store unit, one for the global load/store unit, one common for the branch and the complex integer/multimedia units, and a

Parameter	Value
Threads	1-8
Issue bandwidth	1-8
I- and D-caches	4 MB each
On-chip RAM	32 KB
Instruction fetch units	1-8, up to 8 instrs. each
Load-/store reservation stations	16 entries each
Simple integer reservation station	One common RS with 256 entries
Functional units	10
Result buses	8

single common reservation station for the three integer/multimedia units.

- The reservation stations are set to 16 entries each.
- Also the issue buffer and the reorder buffer are limited to 16 entries per thread (each thread still has a separate issue buffer and reorder buffer). Reorder bandwidth is limited to four instructions per cycle.
- Loads and stores of other threads can pass stores with unavailable memory addresses as in the maximum processor model.
- The on-chip RAM of 32 KB turned out to be essential. Simulations showed an IPC increase of 0.5 to 2.6 depending on the processor configuration by applying the on-chip RAM memory together with a

single or two separate local load/store units. That is even effective for the single threaded models. But we reduced the number of local load/store units to one (instead of two) because it is more realistic.

The simulation results are reported in Fig. 3. A speedup of more than 2.5 over the single-threaded configurations can be reached by multithreading. Increasing the issue bandwidth in the single threaded models from four to eight does not yield any performance gain and increasing the issue bandwidth or the number of threads beyond the 4-threaded 4-issue model reaches only a marginal gain.

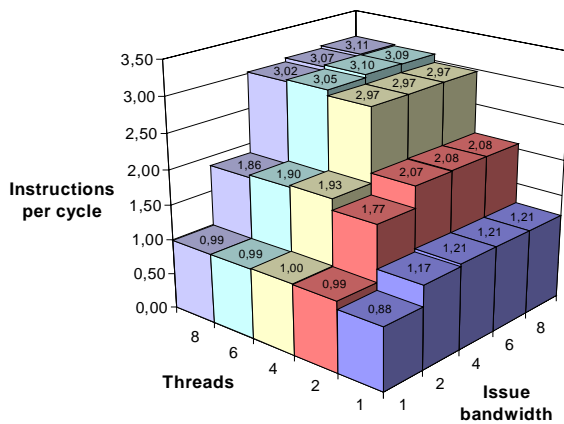


Fig. 3: IPC of the realistic processor with on-chip RAM

To evaluate the tradeoffs in the configuration assumed in Fig 3 several variations were taken into consideration:

Inspecting several thread selection strategies for the 8-8, 6-6, and 4-4 (-threaded, -issue) configurations showed that the simple round-robin strategy is nearly

always as good as or at most 0.2 worse than the more complex strategies (Fig. 4, see [7] for details on the selection strategies). For the realistic processor models we stick to the round-robin strategy because it is the simplest.

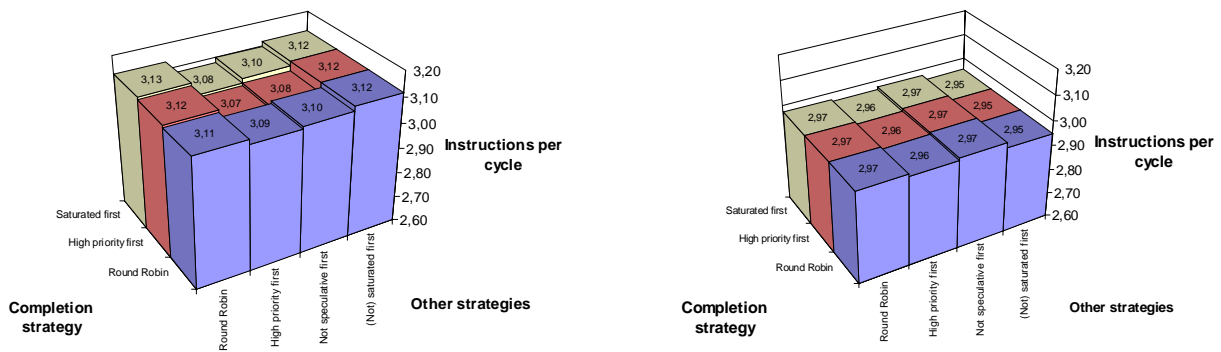


Fig. 4: IPC of thread selection strategies for 8-8 (left) and 4-4 (right) realistic processor configurations

Inspection of I- and D-cache sizes showed that 4 KB or 8 KB are too small. Cache sizes of 32 or 64 KB show only a small performance increase (e.g. Fig. 5 shows an IPC of 3.08 for the 16 KB I- and D-caches versus 3.15

for the 64 KB I- and D-caches in the 8-threaded 8-issue model). In our realistic models we decided to simulate 32 KB I- and D-caches (16 KB could also be used).

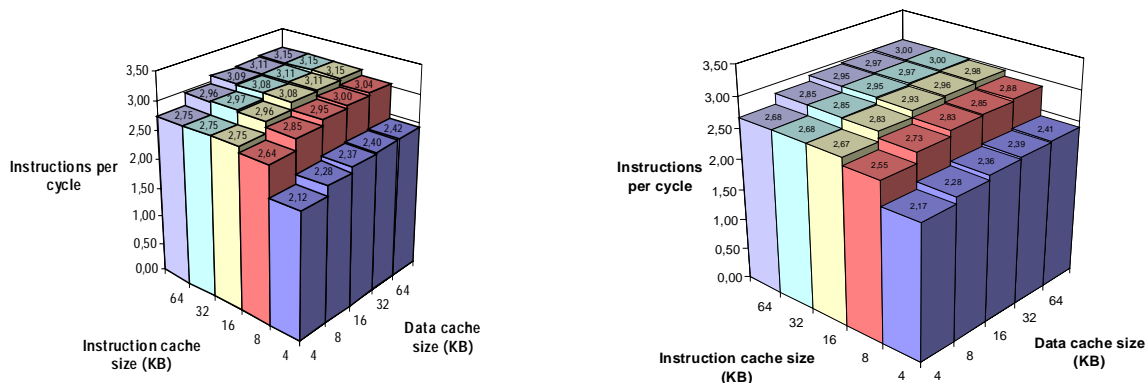


Fig. 5: IPC of data cache size variations for 8-8 (left) and 4-4 (right) realistic processor configurations

Another investigation concerned the D-cache structure. In principle there are two kinds of memory accesses: such that show temporal locality by accessing the same memory address several times within a short time period, and such that have spatial locality by accessing data stored in contiguous memory addresses. Data with spatial locality are only accessed once or only a few times. Such accesses cause cache replacements that may displace data with temporal locality from the D-cache. To solve this problem we simulated a partitioning of the on-chip D-cache in two separate parts: a persistence D-cache for data that exhibits temporal locality, and a streaming D-cache for the data with spatial locality. Data has been split among the now three types of on-chip data memory according to their usage during the handcoding of the MPEG-2 decompression algorithm. Small sets of data that are frequently accessed and modified are stored inside the local RAM memory. This is the case for all interim storage of block data and motion information. Sparse tables that are almost exclusively accessed by read operations are stored in the main memory area buffered by the persistence D-cache. This is the case for the Huffman decoding tables which are repeatedly accessed by the input parser thread. The

reference frame data, which is accessed by the macroblock threads and by the image display thread, is stored in the main memory area buffered by the streaming D-cache.

The simulation results showed no performance increase by our persistence/streaming D-cache model. In fact, the performance was in all configurations about 0.4 to 0.5 lower than the realistic processor performances reported in Fig. 3.

Another investigation concerned the D-cache replacement strategy within a single D-cache. We simulated several alternative cache-replacement strategies with the aim to impede or prevent replacement of the parser tables from the cache (see Fig. 6):

- The conventional round-robin, random selection, and LRU strategies (strategies 0, 1, and 2),
- an instruction-based strategy that allows all threads to access the whole D-cache, but restricts access to cache locations for specific instructions (strategy 3),
- a thread-based strategy aimed to avoid thrashing when different threads access cache-lines that mapped to the same cache location (strategy 4).
- Strategy 5 combines strategies 3 and 4.

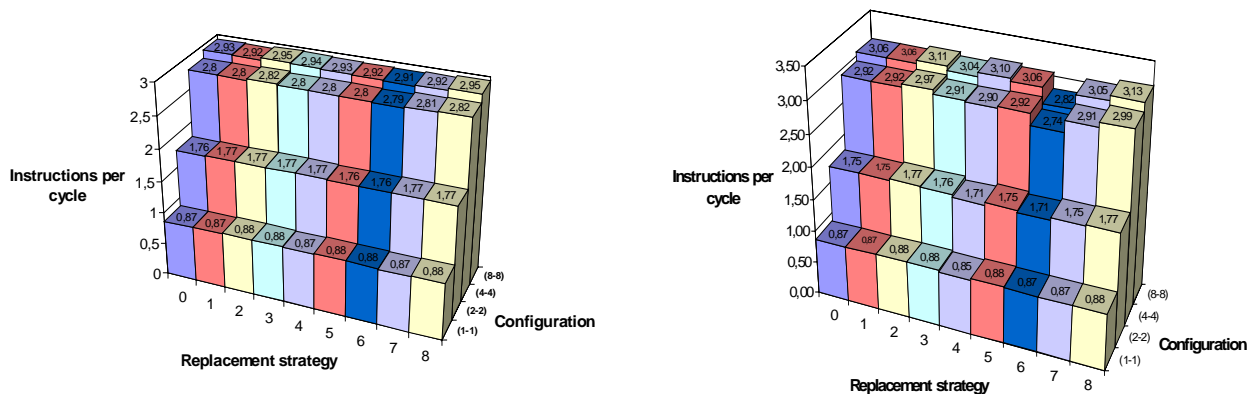


Fig. 6: IPC of various cache replacement strategies for realistic processor configurations assuming 2-way (shown left) and 8-way (shown right) associativity

Three more strategies are based on the idea of a prioritized cache block selection:

- Strategy 6 reserves specific cache locations for threads with high priority. We selected the parser thread as high priority thread to avoid dislocation of the parser tables by data of other threads. However, the overall cache size is limited for the other threads and parts of the cache may be unusable.
- Another idea is a preferred replacement of cache lines of threads with lower priority. The combination with the LRU-strategy leads to different aging speeds of the cache lines (strategy 7).
- Strategy 8 replaces cache lines of low priority threads with a probability twice as high as for the high priority thread.

Simulations of these strategies assuming 2-, 4-, and 8-way set-associative D-caches show little performance differences (Fig. 6 shows 2- and 8-way set-associative D-cache performance; 4-way is in between the 2- and 8-way models). Only strategy 6 shows a slight performance degradation, whereas the LRU-based strategies proved to be best. Our conclusion is that implementation of the more complex cache replacement strategies is hardly worthwhile at least when assuming 2- to 8-way set associative caches. The excellent latency hiding performed by the SMT feature of the processor models allows little room for IPC improvement by applying sophisticated cache strategies. We stick with our realistic processor models to a single 4-way set-associative D-cache with a simple LRU strategy.

Of some success is the technique of speculative data prefetch. Based upon previous accesses with temporal locality, the stride and direction of accessed addresses is stored and an automatic prefetch 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 distinguishes the states “undirected access”, “likely forward”, “forward”, “likely backward”, and “backward”. If the issue bandwidth is at least four, already the single-threaded and the 2-threaded processor configurations profit slightly with the highest profit of 0.13 for the 6-threaded 8-issue model and a profit of 0.06 for the 4-threaded 4-issue case over the values given in Fig. 3. The cache hit rate is improved from 89% up to 96% - 98%.

5 Conclusions

We simulated various SMT multimedia processor models using a hand-coded multithreaded MPEG-2 video decompression algorithm as workload.

The simulations of a realistic processor model showed the steepest performance increases for the 4-issue model from the single-threaded (IPC of 1.21) to the two-threaded (IPC of 2.07) and to the 4-threaded (IPC of 2.97) cases. We therefore suggest the 2-threaded 4-issue or 4-threaded 4-issue processor configurations as realistic next generation processors. A further increase of the issue bandwidth above four will yield none or only a small performance increase. 32 KB code and 32 KB data caches are enough, complex cache strategies yield only a marginal gain because of the excellent latency hiding provided by the simultaneous multithreading technique. 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] J. Silc, B. Robic and Th. Ungerer, *Processor Architecture - From Dataflow to Superscalar and Beyond*, Springer-Verlag, Berlin, Heidelberg, New York, 1999.
- [2] D.M. Tullsen, S. J. Eggers, H. M. Levy, J. L. Lo and R. L. Stamm, Exploiting Choice: Instruction Fetch And Issue on an Implementable Simultaneous Multithreading Processor, *Proc. 23rd Annual International Symposium On Computer Architecture*, Philadelphia, May 1996, pp. 191-202.
- [3] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy and S. S. Parekt, An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors, *Proc. 25th Annual International Symposium On Computer Architecture*, Barcelona, Spain, June 27 – July 1, 1998, pp. 39-50.
- [4] J. P. Wittenburg, G. Meyer, P. Pirsch, Adapting and extending Simultaneous Multithreading for High Performance Video Signal Processing Applications, *Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC99, in connection with HPCA-5 conference*, Orlando, Jan. 9, 1999.
- [5] M. Pontius, N. Bagherzadeh, Multithreaded Extensions Enhance Multimedia Performance, *Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC99, in connection with HPCA-5 conference*, Orlando, Jan. 9, 1999.
- [6] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems, *MICRO-30*, Research Triangle Park, Dec. 1-3, 1997, pp. 330-335.
- [7] H. Oehring, U. Sigmund, Th. Ungerer, Simultaneous Multithreading and Multimedia, *Workshop on Multithreaded Execution, Architecture and Compilation, MTEAC99, in connection with HPCA-5 conference*, Orlando, Jan. 9, 1999.