DLP + TLP Processors for the Next Generation of Media Workloads

Jesus Corbal, Roger Espasa and Mateo Valero
Departament d'Arquitectura de Computadors,
Universitat Politècnica de Catalunya–Barcelona, Spain*

Abstract

Future media workloads will require about two levels of magnitude the performance achieved by current general purpose processors. High uni-threaded performance will be needed to accomplish real-time constraints together with huge computational throughput, as next generation of media workloads will be eminently multithreaded (MPEG-4/MPEG-7). In order to fulfill the challenge of providing both good uni-threaded performance and throughput, we propose to join the simultaneous multithreading execution paradigm (SMT) together with the ability to execute media-oriented streaming μ-SIMD instructions.

This paper evaluates the performance of two different aggressive SMT processors: one with conventional μ-SIMD extensions (such as MMX) and one with longer streaming vector μ-SIMD extensions. We will show that future media workloads are, in fact, dominated by the scalar performance. The combination of SMT plus streaming vector μ-SIMD helps alleviate the performance bottleneck of the integer unit. SMT allows "hiding" vector execution underneath integer execution by overlapping the two types of computation, while the streaming vector μ-SIMD reduces the pressure on issue width and fetch bandwidth, and provides a powerful mechanism to tolerate latency that allows to implement smart decoupled cache hierarchies.

1 Introduction

Media applications will become one of the most demanding types of workloads in the near future. Standards such as MPEG-4 or MPEG-7 will be eminently multithreaded, and a commodity PC will face the need to execute several media streams (encoders, decoders, 3D processing) concurrently. Therefore, the design of future media architectures will have to take into account both the tight real-time requirements of mono-threaded applications and the need to provide high throughput to run multiple tasks simultaneously.

It seems unlikely that current generation microprocessors will be able to meet these requirements in the future if we only scale them in the traditional way (i.e, by adding more functional units and by increasing issue width). We believe that future media workloads intrinsically require some form of on-chip parallel processing if we want to succeed in delivering the required performance. Therefore, architectures able to exploit thread level parallelism, such as Simultaneous Multithreaded Processors (SMT) or Chip Multiprocessors (CMP), appear as the architectures of choice to provide the desired throughput. Of course, in order to deal with the high performance requirements of the kernels of such media workloads we will still need special purpose instructions that can exploit the data-level parallelism available. Thus, μ-SIMD processing is a suitable way of improving uni-threaded performance for media kernels with a modest investment in hardware.

This paper advocates the combination of SMT and μ-SIMD extensions to achieve the performance required for future media workloads. We will evaluate the performance of two different aggressive SMT processors: one with conventional μ-SIMD extensions (such as MMX) and one with longer streaming vector μ-SIMD extensions (our MOM ISA extension).

We will demonstrate that SMT execution and streaming vector μ-SIMD instructions combine very well together since:

- The SMT execution allows mixing scalar and streaming μ-SIMD instructions in an efficient way
- The latency tolerance properties of the streaming μ-SIMD ISA enable the use of decoupled cache hierarchies that avoid the typical cache degradation experienced by multiple threads when running on a SMT

Comparing to a baseline consisting of a plain out-of-order superscalar with multimedia extensions, SMT+MMX yields a 2.1X speedup and SMT+MOM achieves a 3.3X speedup.

2 Multimedia workload trends

In this section we will describe the evolution of media codes and will analyze their main characteristics. We will

*This work has been supported by the Ministry of Education of Spain under contract CICYT TIC98-0511OC02-01 and by the CEPBA. We would like to thank the reviewers of this paper for their useful comments.

discuss some common misconceptions about media workloads that will justify why a combined SMT plus SIMD architecture appears as a suitable alternative for running next generation of media workloads.

From kernels to real programs

When studying multimedia algorithms, or, better, multimedia kernels, most authors agree that their most relevant characteristics are the following [1, 2]:

- Small data type sizes
- Computationally intensive small tight loops
- Large amounts of Data Level Parallelism
- Stream-like patterns, low data locality
- Large memory bandwidth requirements

Most major vendors of high performance processors have realized the importance of media workloads and have extended their ISA's with new instructions targeted at exploiting data-level parallelism over small data types (μ-SIMD parallelism) [3, 4, 5, 6, 7, 8]. These new instructions are generally complemented with stream prefetching instructions in an attempt to alleviate the memory latency difficulties exposed by low-data locality, streaming kernels [5, 9, 10].

However, studying kernels in isolation can be very misleading, since what is true for basic media kernels, does not necessarily apply to complete programs.

A typical media program is composed of a set of kernels that process data in a stream-like fashion and protocol related overhead (table look-ups, header processing, non-vectorizable coding) very similar to what we can find in a typical SPECint benchmark. Since the kernels are repeatedly invoked on sets of related data, their behavior as a group can be very different from their isolated behavior. For example, while at the kernel level one usually encounters low-locality stream-like memory patterns, there is usually high locality at the algorithm level.

Researchers creating benchmarks from raw media kernels usually wrap them in long running loops so that measurements can be easily taken. However, repeating a kernel many times on different data exacerbates its stream-like behavior, which may not be a realistic scenario. As reported in [11], complete media programs characteristics fall somewhere in between raw DLP media kernels and conventional non-numerical applications.

As a result, media-oriented architecture designers should be aware of the following treats that characterize full multimedia programs:

- Computationally intensive small tight loops + integer-like code (protocol overhead)
- Large amounts of DLP in a restricted portion of the execution + restricted ILP in the rest

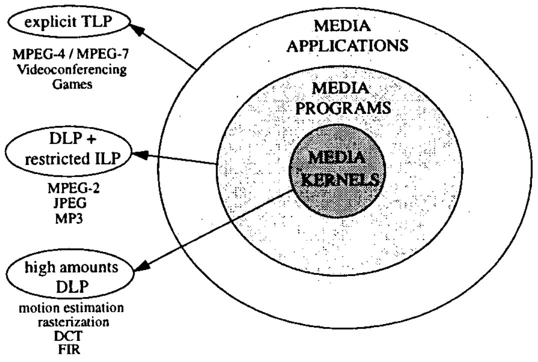

Figure 1. From kernels to media programs, from programs to real applications.

- Stream-like patterns at kernel level but high locality at algorithm level
- Memory bandwidth is the main bottleneck but ILP is also a concern

It follows from this set of characteristics that architectures strictly focused on exploiting the data level parallelism available at kernel level will fail to deliver the expected performance due to Amdahl's law.

From real programs to future media applications

Future media workloads are not expected to change radically the basic multimedia algorithms. Rather, given the large number of different media sources that can be sent over the Internet, the tendency is to focus on joining heterogeneous media streams into unified protocols. These new applications will pay special attention to managing and maximizing the efficiency of compression and/or encryption by extracting uncorrelated media contents from the same source and applying the best media processing for each of them. Additionally, media programs will no longer be monolithic applications. Interactivity will be applied across the board so that different input/output media streams will execute and communicate among themselves concurrently.

The best example of this tendency is the MPEG-4 standard [12]. MPEG-4 is a new ISO/IEC protocol that uses an object-based approach to describe and compose interactive audiovisual scenes. Uncorrelated objects are coded, encrypted and transmitted separately in order to be composed again at reception. These objects may include digital video (MPEG-2), still image, audio, speech and even audio synthesis or 3D-graphics. Several powerful transformations can be performed over every object in order to compose each of them into the same audiovisual scene thanks to a a higher layer of the protocol.

Therefore, future media applications will add several characteristics not contemplated by the current research literature. Dealing with multiple concurrent media streams means that we have high levels of coarse level parallelism (TLP) and that throughput is now also an issue (together with uni-threaded real time requirement of each source). Additionally, an extra layer of protocol means more hard-to-vectorize overhead that may further counterbalance the DLP-only nature of multimedia kernels.

3 Proposed Architecture

The need for coarse grain thread level parallelism comes at a very appropriated moment. Coincidentally, several vendors targeting commercial workloads (OLTP, Web serving and databases in general) have started to focus on new parallel architectures prepared to deal with the abundant, explicit heterogeneous thread-level parallelism that this kind of programs are characterized by.

Looking at recent announcements, two are the most important architectural alternatives: CMPs, or Chip Multi-Processors (Power-4 [13], Piranha [14]), and SMT processors (Alpha 21464 [15]). The first alternative, a CMP, is based on joining together several simple processors on a single die, communicated through a conventional cache hierarchy. The second alternative is based on executing independent flows of execution (threads) concurrently on a (typically) highly aggressive superscalar processor.

Our claim is that this kind of architectures are also appropriated for future media applications. In the same vein that OLTP-like applications, throughput is the major concern in workloads involving several concurrent media sources (once real-time requirements have been met, of course), and both CMP and SMT architectures are good alternatives to meet these throughput requirements. Whether one alternative is better or not is still a matter of controversy: SMT allows a better usage of the available resources while CMP does not have the traditional implementation problems of aggressive out-of-order architectures

From the point of view of overall performance, we believe that SMT processors are specially well suited for the characteristics of media workloads due to the ability of providing moderate performance even in serial fragments of code or with low number of threads (minimizing the impact of Amdahl's Law). In this section we are going to propose a SMT processor with the inclusion of smart media ISA extensions as the architecture of choice for future media desktop systems.

Baseline Processor

Our SMT processor is built around a common out-of-order superscalar processor, as proposed in [16]. As shown in figure 2, our proposed architecture closely resembles a 8-way version of a MIPS R10000, able to fetch up to 8 instructions per cycle. Instructions decoded and renamed are distributed by the dispatch logic to the appropriate instruction queue, which can read from its own dedicated register file. Instructions within every queue may issue out-of-order and a graduation window is in charge of retiring instructions in-order to maintain the appearance of sequential execution.

SMT extension

The basic superscalar architecture has been enhanced following [16, 17] to support Simultaneous Multithreading (SMT). In order for the processor to be able to execute multiple threads concurrently, minor changes are needed for three of the stages of the pipeline: fetch, decode and commit. The fetch engine is able to select up to two groups of 4 instructions per cycle out of the pool of available threads (provided they are not stalled under an I-cache miss or a branch misspredicton). For the initial evaluations, the fetch selection strategy is a classic round robin policy.

As proposed in [16], all threads share a common register pool. The decode engine is able to rename instructions from different threads using a per-thread renaming table and a shared common free register pool. Inside the execution queues no additional logic is required to handle instructions from different threads as renaming provides an easy mechanism to avoid false dependences. Some additional logic is required in the graduation window in order to allow per-thread retirements, as well as a mechanism to perform per-thread instruction flush in case of miss-speculation.

SIMD Extensions

In spite of the explicit thread level parallelism available in future media applications, specific architecture innovations are still needed in order to fulfill the real-time requirement of single media streams. Simultaneous multithreading might provide good overall computational throughput, but, unfortunately, cannot guarantee that, for instance, the frame rate constraints of a MPEG-2 video stream are met. As discussed in previous sections, we believe that new μ-SIMD extensions are necessary to meet the single thread performance requirements of highly demanding multimedia kernels.

We have enhanced our basic SMT core with a multimedia instruction queue, its corresponding SIMD register file and two independent media functional units. Two different sets of multimedia extensions will be evaluated: a μ-SIMD instruction set that resembles the Intel SSE [9] extension and our own streaming-SIMD instruction set, named MOM [11]. Despite differences in their instruction semantics, both use a similar overall architecture.

For the MMX-like instruction set, we have implemented an approximation of SSE [9] integer opcodes with 67 instructions and 32 logical registers (as opposed to 8). We have added some extra features, such as new reduction oper-

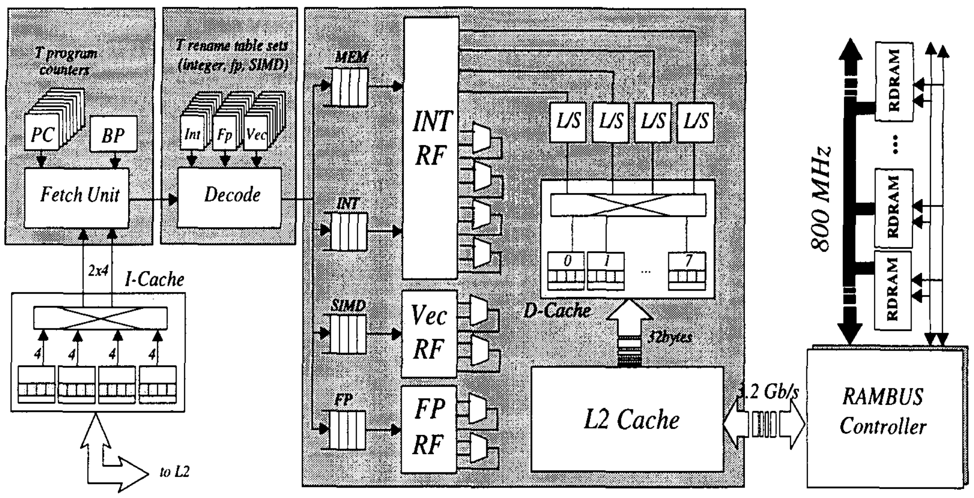

Figure 2. Our basic model of a SMT processor.

ations and multiple source registers, not present in the original SSE.

The MOM instruction set was introduced in [11] and combines the advantages of typical μ-SIMD instructions with the parallelism offered by conventional vector ISAs. By exploiting two different dimensions of parallelism (parallel loops), MOM is able to generate stream instructions that work on up to 16 conventional MMX-like registers and fuse into a single opcode the equivalent of 16 MMX-like instructions. Results presented in [11] demonstrated that this kind of μ-SIMD streams are very well adapted to the characteristics of most media kernels. MOM is loosely based upon the MDMX multimedia extension set from MIPS [4]. MOM has 121 different opcodes and 16 logical stream μ-SIMD registers (each composed of 16 MMX-like registers). In order to improve efficiency in reduction operations, we have also included 2 logical *packed accumulators* of 192 bits. These accumulators allow performing reduction operations over a whole μ-SIMD stream using a single packed accumulator with high efficiency. Finally, our streaming μ-SIMD architecture has one *stream length* register (renamed through the integer register pool) that allows to determine the real length of each stream register (out of 16).

Figure 3 presents a comparison between conventional MMX-like instructions and our MOM stream instructions. A MMX-like instruction is characterized by two parameters: the size of the μ-SIMD register (64-bits typically) and the number of packed sub-word elements (dependent on the size of the sub-word elements). A stream μ-SIMD instruction adds two more parameters: the *Stream Length* and the *Stride*. The *Stream Length* is the number of MMX-like μ-SIMD registers over which the same instruction is going to be executed (up to 16). The *Stride* determines, for stream memory instructions, the distance in memory positions between consecutive μ-SIMD registers inside the same stream.

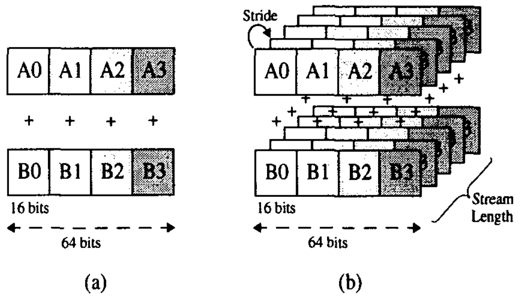

Figure 3. Comparison between (a) a conventional MMX-like μ-SIMD instruction and (b) a Stream μ-SIMD instruction.

The stride feature is very powerful for multimedia (specially for image/video processing), as it allows to work over small sparse matrices of data.

Architectural Parameters

Our basic processor configuration is able to issue up to 4 integer instructions, up to 4 memory instructions (either loads or stores) and up to 4 floating point instructions per cycle. Additionally, the SMT+MMX processor is able to issue up to two different MMX-like instructions per cycle. On the other hand, our SMT+MOM processor has only one single media functional unit of width 2 (that is, we have two parallel vector pipes so that up to two μ-SIMD sub-instructions can be executed every cycle from the same stream). As a result, in contrast with the conventional MMX-like version, the SMT+MOM processor only requires an issue width of 1 for the SIMD queue.

	MMX				MOM			
	1 th	2 th	4 th	8 th	1 th	2 th	4 th	8 th
INT queue	32	32	64	64	32	32	64	64
FP queue	16	16	32	32	16	16	32	32
VEC queue	16	16	32	32	8	8	16	16
L/S queue	16	16	32	32	16	16	32	32
INT reg	76	102	172	320	76	102	172	320
FP reg	40	80	148	280	40	80	148	280
SIMD reg	64	82	152	304	32	51	120	146
ACC reg	-	-	-	-	4	8	16	24
Fetch policy	1x8	2x4	2x4	2x4	1x8	2x4	2x4	2x4

Table 1. Architectural parameters based on number of threads.

Our SMT processor simulator contains a highly detailed memory hierarchy model, where both L1 and L2 cache levels are located on-chip (as in the Alpha 21364 [18]). The L1 cache is a 32 KB, direct mapped, write-through cache with 32-byte lines, interleaved among 8 memory banks. The I-cache is a 64 KB, 2-way set associative cache with 32-byte lines, interleaved among 4 memory banks. The L2 cache is a 1MB, 2-way set associative, write-back cache with 128-byte lines. Both L1 and L2 levels of cache have 8 MSHRs and a 8-depth coalescing write buffers with selective flush policy. L1 and I1 have one cycle of latency while the L2 cache latency is 12 cycles. We have modeled a 128MB *Direct Rambus* main memory system which contains a *DRDRAM* controller driving 8 *Rambus* chips and leveraging up to 3.2 GB/s with a 128-bit wide, bi-directional 200Mhz main bus (feeding a 800MHz processor).

We have done preliminary simulations in order to determine the number of physical registers and the window sizes necessary to achieve reasonable (near saturation) processor performance for 1, 2, 4 and 8 threads. The results can be seen in table 1. Note that the size of the stream μ-SIMD register file can be up to 8 times the size of the MMX register file. However, as already seen in [11], organization in lanes and interleaving of the different elements of the vector register into banks help to decrease radically the overall area without any impact on performance. Further discussion on the impact on cycle time of the large number of registers is beyond the scope of this paper.

4 Workload characterization

In this section we will start by defining our workload, inspired in the MPEG-4 media profiles. Then, using two different ISAs, (Compaq's) *Alpha* extended with μ-SIMD instructions (MMX-style) and *Alpha* extended with our own streaming μ-SIMD instructions, we will present an instruction breakdown of the workload.

4.1 Modeled Workload

Trying to evaluate what will be a future media workload is not easy. Parameters that influence overall application behavior, such as the predominance of each media source, the size of its working set, or the level of protocol overhead, are hard to determine *a priori*. Even already standardized protocols such as MPEG-4 are still slightly ambiguously defined (MPEG-4 was promoted as an ISO standard just in 1999) and it is difficult to obtain reliable, non research-oriented source codes.

Therefore, our methodology will be based on selecting a set of real multimedia programs that approximate the multiprogrammed contents of a full MPEG-4 application. According to the standard, MPEG-4 is composed of four different profiles (or heterogeneous contents):

- MPEG-4 control (BIFS-base scene descriptors)
- MPEG-4 video (MPEG-2)
- MPEG-4 still image (2D and 3D graphics)
- MPEG-4 audio (speech codec, audio synthesis)

We have used programs from the *Mediabench* suite [19] that represent each one of the aforementioned profiles. MPEG-2 encode and decode are examples of the MPEG-4 video profile. The public domain implementation of OpenGL, mesa, and the JPEG coding/encoding match the 3D and 2D MPEG-4 still image profiles. Finally, gsm encode and decode represent the MPEG-4 audio speech profile. The only profile not included in our workload is the MPEG-4 control (that deals with VRML-like compositioning of the sources into the same scene). Note that this last profile would likely have a certain impact on performance, as it would create dependences and synchronization needs between threads. Table 2 describes the set of benchmarks selected for our multiprogrammed workload.

For each program, we identified the most important functions using profiling and manually rewrote the vectorizable ones using both MMX-like instructions and our stream μ-SIMD instructions, by means of our own emulation libraries [11]. We should note that the 3D graphics benchmark (mesa) has not been vectorized because our emulation libraries do not have floating-point μ-SIMD instructions.

4.2 Instruction breakdown

Table 3 shows an instruction breakdown for each of the benchmarks for the two μ-SIMD instruction sets under consideration, MMX and MOM. The last row in the table gives the total number of instructions per benchmark (in millions). Note that, to allow for a meaningful comparison, a MOM μ-SIMD instruction that operates with, say, a stream length of 11, counts as eleven instructions (i.e., each MOM instruction is multiplied by its stream length). The first four

	instances	description	data set	characteristics
mpeg2 encode	1	MPEG2 digital video encoder	mei16v2rec.Y/Cb/Cr	bit stream of four 24-bit colour 352x480 frames in YCbCr format
mpeg2 decode	2	MPEG2 digital video decoder	mei16v2rec.mpg	MPEG2 video stream, four 352x480 frames
jpeg encode	1	JPEG still image encoder	penguin.ppm	ppm file, 24-bit colour 1024x739 image
jpeg decode	1	JPEG still image decoder	penguin.jpg	JPEG file, 1024x739 image
gsm encode	1	GSM 06.10 full-rate speech encoder	clinton.pcm	8Khz sampling rate, 300KB PCM audio stream
gsm decode	1	GSM 06.10 full-rate speech decoder	clinton.gsm	13Kb/s GSM audio stream
mesa	1	OpenGL 3D-graphics synthesis	teapot.ppm	texture mapped version of Utah teapot, 24-bit colour 500x500 ppm file

Table 2. Multi-programmed workload description.

	MPEG2 enc		MPEG2 dec		JPEG enc		JPEG dec		GSM enc		GSM dec		mesa		total	
	mmx	mom	mmx	mom	mmx	mom	mmx	mom	mmx	mom	mmx	mom	mmx	mom	mmx	mom
INT	56	54	58	62	62	68	58	58	78	83	90	90	55	55	62	65
FP	0	0	0	0	0	0	0	0	0	0	0	0	14	14	1	1
MEM	23	36	29	32	19	22	20	20	13	15	9	9	31	31	21	26
VEC	21	10	13	6	19	10	22	22	9	2	1	1	0	0	16	8
#ins	642.7	364.9	69.8	59.8	160.3	135.8	109.4	106.4	177.9	161.3	105.2	105.0	93.8	93.8	1429	1087

Table 3. Instruction breakdown (%) and instruction count (in millions of instructions)

rows present the percentage of each type of instruction, integer arithmetic, floating point, SIMD arithmetic and memory (both scalar and vector memory instructions), in each benchmark.

In sharp contrast with the common belief, table 3 shows that under MMX, our multimedia workload is dominated by the integer pipeline (62% on average). SIMD arithmetic instructions only account for 16% of the overall number of instructions. On top of everything, the workload is characterized for having a very unbalanced distribution of instructions at run-time. Media programs execute typically regions of code with a high percentage of vector instructions and few scalar instructions and other regions of code with no SIMD instructions at all (thus causing severe resource balancing problems).

The stream μ-SIMD paradigm substantially reduces the number of integer instructions (around 20%) and memory instructions (around 7%) when compared to MMX. The reason is a phenomena commonly found in any conventional vector architecture. As every stream μ-SIMD instruction can pack several MMX-like instructions (thus, replacing multiple instances of a loop) there is an elimination of scalar instructions related to the loop control (that is, backward branches, loop indexes or address calculations). Our stream instructions generate all this information automatically, thanks to the information of the *Stream Length* register and the *stride*. There is an even higher reduction of the overall number of vector instructions (62 %). The reason is that our stream ISA can take great advantage of MDMX-like *Packed Accumulators* (as seen in [11]). These accumulators are very useful in reduction sequences, eliminating a high amount of logic overhead.

However, in terms of *relative* percentage, the MOM paradigm does not alleviate the predominance of integer instructions in the instruction mix. Quite the contrary, it slightly increases the percentage of integer instructions. Thus, independently of using MMX or MOM, table 3 clearly shows that the integer pipeline will be the main performance bottleneck within the CPU when executing our approximation of a next generation media workload (ignoring, of course, memory behavior). Therefore, the best we can expect our SMT architecture to do is, at most, hide the execution of all memory and SIMD instructions underneath the execution of the integer instructions of the program.

5 Performance evaluation

In this section, we are going to evaluate the performance of the SMT architecture with the two SIMD ISAs under study. We will present first performance under an ideal memory system and then, we will evaluate the impact of a realistic memory model. Finally, we will study the performance improvements using smart fetch policies and will analyze alternative cache hierarchies in order to alleviate the memory problem.

5.1 Simulation methodology and performance metrics

We have evaluated the performance of the modeled workload using an SMT version of the Jinks simulator [11] with 1, 2, 4 and 8 threads. In order to do so, we selected a random order of the 8 programs: MPEG-2 encoder, GSM decoder, MPEG-2 decoder, GSM encoder, JPEG decoder, JPEG encoder, mesa and MPEG-2 decoder (2^{nd} time). Simulation starts with as many programs concurrently as the number of contexts allowed by the machine. When a program completes, the next program from the list is initiated. In case that no further programs are available, we initiate again selecting programs from the same list from the beginning. This process is repeated until the end of the 8^{th} context. This avoids

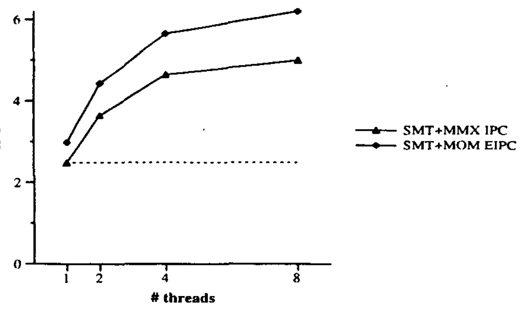

Figure 4. Performance with perfect cache.

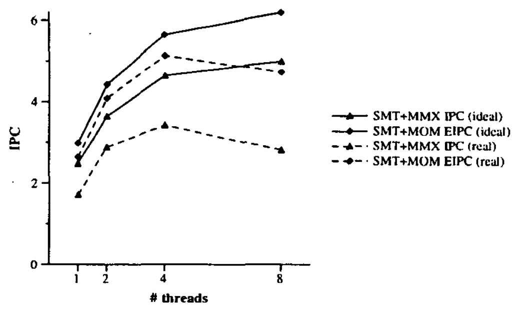

Figure 5. Performance under real memory system.

having fractions of time with less threads than those allowed by the machine. In order to round to 8 programs, the most significant program (MPEG-2 decode) is included twice.

We should note that this methodology gives us a measure of throughput rather than real execution time. We believe that this is the most suitable metric as future media workloads will be characterized by continuous media streams being executed concurrently along the time.

When evaluating the performance of a SMT architecture, we typically use IPC (instructions committed per cycle) as a good indicator of throughput. However, the IPC is not a good measure of performance when comparing different ISAs, as every ISA needs a different number of instructions to execute a given benchmark. Therefore, in this paper we will use a new indicator of performance for the streaming μ-SIMD architecture:

$$EIPC_{MOM} = \frac{instructions_{MMX}}{instructions_{MOM}} \times IPC_{MOM}$$

EIPC stands for Equivalent IPC, and intuitively indicates the IPC a SMT+MMX processor should reach in order to match the performance of the SMT+MOM processor. The ratio between the EIPC of the SMT+MOM architecture and the IPC of the SMT+MMX architecture gives a measure of performance Speed-up.

5.2 Performance with Ideal Memory Systems

Figure 4 shows performance for the two architectures under study with a idealistic memory system (neither cache misses nor bank conflicts). The horizontal dotted line represents the baseline performance of a single thread with MMX instructions. From the results of the figure, we can see that, as we increase the number of threads, SMT+MMX goes from the baseline EIPC of 2.47 up to 5.0 (a speedup of 2.02X). Even better, SMT+MOM goes from an EIPC of 2.98 (20% better than MMX) for a single thread up to 6.19 (a speedup of 2.08X). Overall, SMT+MOM is 2.5 times better than an 8-way superscalar with MMX instructions.

As we will see later, the MOM model achieves even higher relative performance under realistic memory assumptions and when a better mix of scalar and vector instructions is performed.

		1 thread	2 threads	4 thread	8 threads
I Hit Rate	MMX	99.0%	97.8%	96.9%	93.7%
	MOM	98.7%	98.2%	96.6%	93.9%
L1 Hit Rate	MMX	98.7%	97.6%	94.2%	86.8%
	MOM	98.4%	98.1%	96.9%	93.7%
L1 Latency	MMX	1.39	1.59	2.38	6.81
	MOM	1.74	1.86	2.43	4.51

Table 4. Memory performance degradation: I-cache hit rate, L1 hit rate and L1 average memory latency

5.3 Performance under Real Memory

Figure 5 shows performance for the two architectures under study taking into account the effect of the memory system described in section 3. Performance with ideal memory is also presented for comparison purposes.

From the results of the figure we can observe two main phenomena: (a) Increasing the number of threads may provide diminishing returns (performance with 4 threads is higher with 8 threads), and (b) MOM is more robust to the impact of the memory system (MOM exhibits an average performance degradation of a 12 % in comparison to the 30 % of MMX).

In order to understand these two effects, we may look at table 4, where the instruction cache hit rate, the L1 cache hit rate and the average memory latency on L1 are shown. We may observe how, as long as we increase the number of threads that can co-exist in the processor, the hit rate of both instruction and L1 caches decrease, due to the mutual interference between threads. This ends up increasing the memory latency, thus reducing performance.

The higher robustness of MOM against the thread-interference is due to two main reasons: a lower hit rate degradation and a higher memory latency tolerance. A lower hit rate degradation is due to the nature of the stream memory accesses. Since a stream memory reference determines several memory accesses from the same thread, the thread

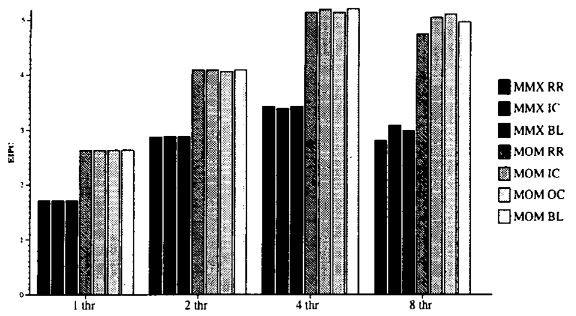

Figure 6. Impact of the different fetch policies.

interference is reduced. Additionally, the stream memory references exhibit a high memory latency tolerance since, as usual with vector memory references, they can amortize the memory latency across the different elements accessed.

Performance of Advanced Fetch Mechanisms

As described in [20], the way we select instructions to be fetched may have a relevant impact over final performance. This is specially true for the stream μ-SIMD architecture, where we found that the conventional round-robin policy was not able to optimally mix scalar and vector instructions (with 8 threads, only a 1% of the MMX execution cycles perform only vector instructions while in MOM, a 4% of the execution cycles perform only vector instructions).

We have evaluated the performance of the following fetch policies:

ROUND-ROBIN (RR) The basic round-robin policy used until now.

ICOUNT (IC) Based on the fetch policy proposed in [20]; priority is given to those threads with the lower number of instructions decoded but not issued.

OCOUNT (OC) - Similar to ICOUNT, but taking into account the information of the *Stream Length* register from the MOM architecture to give lower priority to those threads with the higher number of not-issued operations.

BALANCE (BL) - Focused on mixing scalar and vector instructions. If there are no instructions in the vector pipeline, threads that fetched vector instructions the last time are given priority. In other case, threads that did not fetch any vector instruction the last time are given priority. A round-robin policy is used to chose between threads with the same priority.

Figure 6 shows the performance for all the fetch policies under study for both the SMT+MMX and SMT+MOM architectures. We can observe that the different fetch policies are only effective with a high number of threads (compared to the round-robin), delivering up to a 9 % of performance improvement. Note, however, that, in spite of the fact that performance degradation is smoothed, performance for 4 threads is still higher that performance for 8 threads.

The ICOUNT is the policy that leverages higher performance for the SMT+MMX model, while OCOUNT is the policy that exhibits the best performance for the SMT+MOM model. BALANCE stands as a cost-effective alternative, given the simplicity of its implementation compared, for instance, with the OCOUNT policy.

5.4 Decoupling the Cache Hierarchy

As seen in section 5.3, if we increase the number of threads of the processor, the data locality is reduced due to inter-thread interference. As a result, we incur in higher latency penalties and a loss of bandwidth efficiency. Moreover, as we increase the number of threads, the number of available memory instructions to execute per cycle raises, thus increasing the likelihood of bank collisions (reducing even more the bandwidth efficiency). While the latency penalty impact can be overridden partially by the latency tolerance properties of SMT execution, the loss of effective bandwidth is a problem that directly affects final performance.

In [21] we proposed to bypass vector memory accesses to a higher level cache and to decouple general purpose memory ports into scalar memory ports and vector memory ports (as seen in figure 7). Decoupling scalar and vector memory ports into different levels of the cache hierarchy achieves two goals: (a) we effectively decouple the vector working set from the scalar working set, and (b) we reduce the number of memory ports per level of cache, thus reducing bank contention.

Figure 7 compares the original 4-port cache organization we have just evaluated with a decoupled organization. In the latter configuration, we have 2 memory ports to access scalar elements from the L1 (single-banked and double-pumped as in the Alpha 21264), and 2 memory ports directly connected to L2 used for stream SIMD memory accesses (of course, the L2 still has to talk to the L1 and I1 caches). The L2 has two banks connected to the vector memory ports via a crossbar. Naturally, bypassing the L1 when doing stream accesses can cause coherence problems due to interference between vector and scalar data. Consequently, a coherency protocol based on an exclusive-bit policy is used to deal with this situation [21].

Figure 8 shows the performance of the decoupled cache hierarchy for all the different fetch policies studied so far. The first main conclusion is that the new cache decoupling strategy solves the cache degradation problem: contrary to the data in figure 5, now the 8-thread configuration is better than the 4-thread configuration. Another observation is that the different fetch policies barely provide any performance benefit for the SMT+MMX architecture, while they provide up to a 7 % of performance improvement for the SMT+MOM processor model.

In order to be able to compare the efficiency of the different cache hierarchy strategies, we may look at figure 9, which shows performance results for the three different cases: ideal memory system, conventional memory hierarchy, and the decoupled memory hierarchy. For the

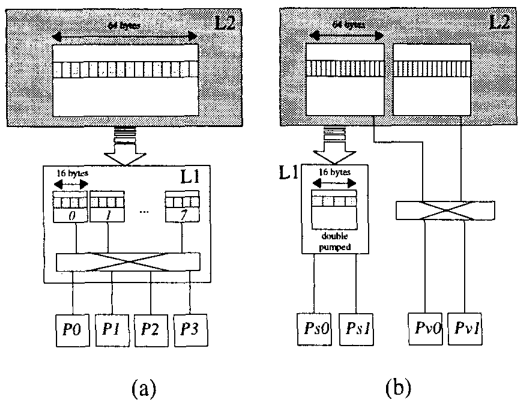

Figure 7. Proposed cache hierarchies: (a) conventional memory ports, (b) decoupled scalar and vector memory ports.

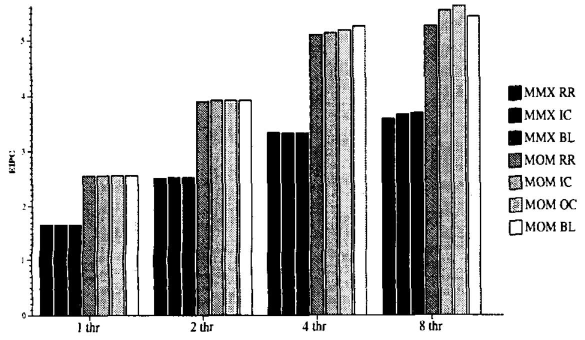

Figure 8. Impact of the different fetch policies under the decoupled cache hierarchy.

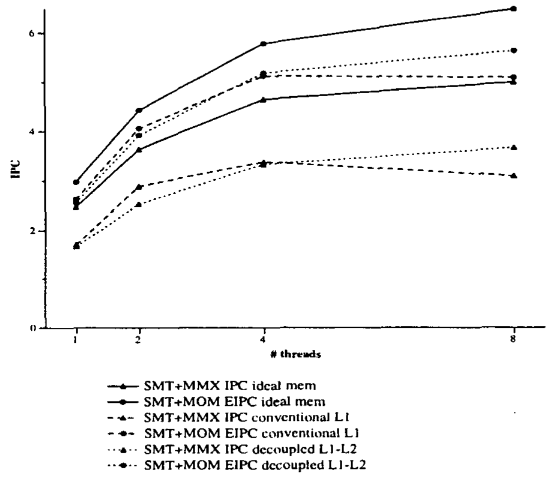

Figure 9. Performance benefits of bypassing L1 on vector memory accesses.

MMX-like architecture, results with the ICOUNT policy are given, while MOM performance results are presented with the OCOUNT policy.

From the results of the figure, we can observe that bypassing the L1 cache is very useful but only if we have a large number of threads, since in that case we are able to tolerate the 12-cycles of L2 latency while taking advantage of the higher effective bandwidth. Our MOM instructions benefit even more from the decoupled hierarchy due to its own additional latency tolerance capabilities. As a result, the MOM+SMT architecture exhibits only a 15% of performance degradation compared with the idealistic memory system, while the SMT+MMX exhibits a 30% of performance degradation (for the 8-thread configuration).

6 Related Work

SIMD execution appears as a natural choice to provide high uni-threaded performance. During the last years, general-purpose designers have been including μ-SIMD (sub-word level SIMD parallelism) extensions to their instruction-set architectures. Intel's media extension family (*MMX* [3],*SSE* [9] and *SSE2* [10]) is perhaps the most widely known set of media-enhanced instructions but almost all of the main general purpose vendors have included one of their own [5, 6, 7, 4, 8]. Authors have also proposed using more classical vector designs or specialized streaming architectures to target multimedia programs [22, 23, 24, 25].

There has been a large body of research regarding SMT architectures [16, 20, 26, 27], most of them specially focused on pure TLP to ILP exploitation. Phenomena like branch missprediction and memory latency tolerance, cache degradation and impact of fetch policies (all of them typical of the SMT paradigm) have been deeply studied.

The contribution of this paper is the claim that future media applications will provide explicit TLP that can be exploited with great benefit by SMT processors with μ-SIMD media extensions. A previous proposal [17] evaluated the performance of a SMT processor with conventional vector enhancements for numerical applications. Nonetheless, we believe that the combination of the two paradigms is more promising for media processing, as numerical codes are dominated by the vector component rather than the scalar one. This paper has shown that ILP exploitation together with an effective mixing of scalar and vector instructions is extremely relevant for media performance.

In [28], a SMT processor with MMX-like extensions is used to evaluate the performance of a parallel version of an MPEG2-decoder. The authors identified as one of the major performance bottlenecks the large serial fragment of non-vectorizable code. Our proposal differs in two main points: (a) on the convenience of exploiting heterogeneous explicit

TLP rather than exploiting TLP over the already vectorized (thus, with high DLP) fragments of code; and (b) on the convenience of using streaming μ-SIMD instructions to increase the latency tolerance (thus allowing smarter memory hierarchies).

Finally, there has been another kind of architecture proposals targeted at future media applications. The M-PIRE processor [29] is an architecture explicitly focused on executing MPEG-4 applications. Its basis of implementation is the partition of the processor into independent programmable units, each of them optimized for a certain class of MPEG-4 algorithm. With a similar philosophy, the Sony's PSX2 Emotion Engine [30] uses independent vector/micro-coded units able to exploit SIMD parallelism concurrently with the execution of the main integer core.

7 Summary

In this paper we have studied and evaluated the performance of an efficient architecture for the next generation of media workloads. We have shown that in order to match the requirements of future standards such as MPEG-4, both DLP and TLP can be exploited efficiently, and we have proposed SMT processors enhanced with μ-SIMD extensions as a suitable alternative.

We have evaluated two different μ-SIMD alternatives (a MMX-like extension and a stream, vector-like μ-SIMD ISA) and have shown the advantages of stream-oriented μ-SIMD alternatives such as MOM.

We have seen that while the SMT capabilities allow to hide vector execution behind integer execution, (thus minimizing the impact of Amdahl's Law) the latency tolerance properties of MOM memory streams allow to introduce smarter cache hierarchies that help alleviate the cache performance degradation associated with the inter-thread interference. As a result, while SMT provides a maximum speed-up of 2.1X for the MMX processor (with a 30% of performance degradation compared with idealistic memory performance), the MOM processor achieves a performance improvement of 3.3X (compared with the performance of a uni-threaded MMX model), suffering from only a 15% of performance degradation from the impact of a realistic memory model.

References

[1] K. Diefendorff and P.K. Dubey. How multimedia workloads will change processor design. *IEEE Micro*, Sep 1997.

[2] T.M. Conte et. al. Challenges to combine general-purpose and multimedia processors. *IEEE Computer*, Dec 1997.

[3] A. Peleg and U. Weiser. MMX technology extension to the INTEL architecture. *IEEE Micro*, August 1996.

[4] Mips extension for digital media with 3D. Technical Report http://www.mips.com, MIPS technologies, Inc., 1997.

[5] K. Diefendorff, P.K. Dubey, et. al. Altivec extension to powerPC accelerates media processing. *IEEE Micro*, March-April 2000.

[6] 3DNow! technology manual. http://www.amd.com, Advanced Micro Devices, Inc., 1999.

[7] M. Tremblay, J.M. O'Connor, V. Narayanan, and L. He. VIS speeds new media processing. *IEEE Micro*, August 1996.

[8] R. Weiss J. Hicks. Motion video instructions (MVI). http://www.alphalinux.org/docs/MVI-full.html, Compaq, 1999.

[9] Pentium III processor: Developer's manual. http://developer.intel.com/design/PentiumIII, INTEL, 1999.

[10] http://developer.intel.com/design/processor/index.htm. Willamette Architecture Software Developer Manuals. Intel, 2000.

[11] J. Corbal, R. Espasa, and M. Valero. Exploiting a new level of DLP in multimedia applications. *MICRO*, 1999.

[12] R. Koenen. MPEG-4, multimedia for our time. *IEEE Spectrum*, February 1999.

[13] K. Diefendorff. Power4 Focuses on Memory Bandwidth. *MicroProcessor Report*, October 1999.

[14] L.A. Barroso, K. Gharachorloo, et. al. S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In *ISCA'00*, June 2000.

[15] J. Emer. Simultaneous Multithreading: Multiplying Alpha's Performance. In *Presentation at the MicroProcessor Forum'99*, October 1999.

[16] D.M.Tullsen, S.J.Eggers, and H.M.Levy. Simultameous multithreading: Maximizing on-chip parallelism. *ISCA-22*, June 1995.

[17] R. Espasa and M. Valero. Exploiting Instruction- and Data- Level Parallelism. *IEEE Micro*, Sept/Oct 1997.

[18] Peter Bannon. Alpha 21364: A Scalable Single-chip SMP. http://www.digital.com/alphaoem/microprocessorforum.htm, Compaq Computer Corporation, 1998.

[19] C. Lee, M. Potkonjak, and W.H. Magione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communication systems. *MICRO 30*, 1997.

[20] D.M.Tullsen, S.J.Eggers, J.S.Emer, et al. Exploiting choice: Instruction fetch and issue on an implementable simulataneous multithreading processor. *ISCA-23*, May 1996.

[21] F. Quintana, J. Corbal, R. Espasa, and M. Valero. Adding a vector unit on a superscalar processor. *ICS*, June 1999

[22] W. J. Dally. Tomorrow's computing engines (Keynote Speech). In *HPCA-4*, February 1998.

[23] K. Asanovic. Vector microprocessors. Phd thesis, University of California at Berkeley, 1998.

[24] C. G. Lee and M. G. Stoodley. Simple Vector Microprocessors for Multimedia Applications. In *MICRO 31*, December 1998.

[25] C. Kozyrakis and D. Patterson. A new direction for computer architecture research. *IEEE Computer*, November 1998.

[26] W. Yamamoto, M.J. Serrano, A.R. Talcott, R.C. Wood, and M. Nemirovsky. Performance estimation of multistreamed, superscalar processors. *27th Hawaii International Conference on System Sciences*, January 1994.

[27] S. Hily and A. Seznec. Out-of-order execution may not be cost effective on processors featuring simultaneous multithreading. *HPCA*, Jan 1999.

[28] H. Oehring, U. Sigmund, and T. Ungerer. MPEG-2 video decompression on simultaneous multithreaded multimedia processors. *PACT'99*, October 1999.

[29] J. Kneip, B. Schmale, and H. Moller. Applying and implementing the MPEG-4 multimedia standard. *IEEE Micro*, Nov-Dec 1999.

[30] A. Kunimatsu, N. Ide, and T. Sato et. al. Vector unit architecture for emotion synthesis. *IEEE Micro*, March-April 2000.