14th European Signal Processing Conference (EUSIPCO 2006), Florence, Italy, September 4-8, 2006, copyright by EURASIP

CURRENT AND FUTURE TRENDS IN EMBEDDED VLIW MICROPROCESSORS APPLIED TO MULTIMEDIA AND SIGNAL PROCESSING

Giuseppe Desoli, Thierry Strudel, Jean-Philippe Cousin, Kaushik Saha

STMicroelectronics/Advanced System Technology
Via Cantonale 16E, 6928, Manno, Switzerland
12, rue Jules Horowitz, 38019 Grenoble Cedex, France
Plot #1, Knowledge Park III, Greater Noida, India
email: {giuseppe.desoli,thierry.strudel,jean-philippe.cousin,kaushik.saha}@st.com
web: www.st.com

ABSTRACT

Although Very Long Instruction Word (VLIW) processors mix of performance, power consumption, flexibility and cost is a very good match for embedded systems in general and multimedia streaming ones in particular; they might be adversely exposed to increasing memory latencies, code size bloat and to some extent performance scalability with increasing issue width. This paper presents two extensions for such VLIW micros that have a large potential impact when applied to the highly competitive market of multimedia consumer applications and more recently streaming: Symmetric Multi Processor (SMP) cache coherency and multithreading; we present a quick summary of those developments carried out by STMicroelectronics within the framework of the ST200 family of embedded microprocessor and preliminary results obtained from their use in selected video and audio applications.

1. INTRODUCTION

This paper focuses on two major architecture extensions: symmetric multiprocessing (SMP) and simultaneous multithreading (SMT) that, although well established in the state-of-the-art for workstation class microprocessors and supercomputers alike, presented a number of challenges when applied to the domain of embedded multimedia microprocessor cores and specifically to the ones that rely on the VLIW architecture to exploit instruction level parallelism (ILP).
In the last few years, due to new design spaces opened by advancements in silicon technologies and rapid scaling up of transistor budgets, the perceived disadvantages of VLIWs have diminished in importance, because of that and also thanks to architectural enhancements, VLIW architectures are growing in popularity, particularly in the embedded market, where the metrics used to drive microprocessor evolution are tightly coupled with performance, cost, area and power consumption trade-offs. Embedded VLIW products are available from several vendors, including Fujitsu, the ST231 from STMicroelectronics, the Jazz DSP from Improv Systems, Silicon Hive, Texas Instruments TMS-C6 and the Philips trimedia series to name some.
One of the problems is a lack of robustness with respect to pipeline stalls induced by various external events, such stalls are in general associated with memory requests and traffic congestion on a typical embedded system-on-chip (SoC). The pipeline stalls can't in general be partially avoided in VLIWs, like in the case of superscalar processors, because they lack the complex logic that allows superscalars to reorder instructions and issue them in program order. This inability, although a direct result of the great simplification in implementation complexity at the pipeline level, can adversely impact performance for certain classes of applications where the cache access patterns can't be sufficiently predicted and for which data prefetching, when such a feature is available, doesn't help either.
Newer multimedia application requirements, especially in combination with data streaming over all kinds of communication channels might introduce even more uncertainty to system's data availability; this is associated with variable and simultaneous workloads being a mix of control functions, such as protocol stacks, synchronization, error concealment and correction, data-rate control, etc. and the more traditional signal processing data-crunching. Modern multimedia streaming application, can be implemented with the support of programming models and run-time environments exploiting multiple threads of execution; this naturally stemming from the heterogeneous set of different functions and the need to handle streams of different nature and at different data-rates. As a consequence it's often the case that interactions between different software entities create more complex memory access patterns, and also expose higher network latencies to the processors, further reducing their overall performance when suitable countermeasures are not available. To address these problems few remedies exist, from more intelligent data prefetching to various forms of scoreboarding that all tend to add too much complexity.
The other potential problem is scalability; VLIWs have been built with very large instruction words and number of functional units, however most embedded microprocessors are limited in terms of transistor budget and pipeline frequency when compared to off-the-shelf general purpose processors. In practice for implementation reasons, is not frequent to see offerings with more than 16 functional units (FUs) and/or issue widths higher than 8 instructions per cycles. Here we report on two techniques that although not new in terms of general computer architecture, have to our knowledge, never being applied to an embedded general purpose VLIW processor before; the first addresses the problem of increased robustness to memory/network latencies, application workloads and simultaneous threads interactions by way of providing the capability of simultaneously running multiple threads of execution, a technique commonly referred to as simultaneous multi threading (SMT); the other describes an efficient implementation of a cache coherency protocol for an embedded VLIW multimedia processor that keeps into account as primary criteria, cost, power and latency. In the following we will provide the description of the design of such features into the ST200 family of VLIW embedded microprocessors core, originally jointly designed by STMicroelectronics and Hewlett-Packard.
First we describe the ST231MT multithreaded microprocessor core with a brief description of prior state-of-the-art, then the description of the ST231MP symmetric multiprocessing capable core. Section 3 describes parallel implementation of video and audio decoders followed by results obtained on few representative application work-

loads both for independent applications mixes and parallel ones; finally we provide some conclusion and future work.

2. MULTI-THREADING AND VLIW

Simultaneous multithreading (SMT) supports multiple independent threads of execution with the aims of better utilizing the datapath resources, partially hide latencies and increase throughput in general. Early attempts date back to 1950's with machines from Bull Gamma and Honeywell; the first commercial SMT machines appeared in 1978 and 1980 and the technique has been deployed in alternating phases of popularity and abeyance throughout modern days. It's worth mentioning that network processors have too frequently exploited some form of multithreading given the favourable nature and level of concurrency of the networking applications. A thorough timeline is available in [1].

Often SMT execution models have been designed in superscalar microprocessors that already support instruction reordering by way of register renaming techniques; for which the additional complexity is marginal and instructions are buffered and dependencies analysed prior to be dispatched to the functional units. Exposing the instruction scheduler to multiple threads is relatively simple then, as instructions from different threads are independent.

The register renaming logic and shadow registers can be leveraged to avoid full replication of each thread's register; only a fractional increase of the number of shadow registers could be required depending from the number of HT supported. On the other end, VLIW cores rely on compilers for static instruction scheduling and lack an instruction scheduler and register renaming logic. For this reason we choose to adopt a multithreaded flavour limited to a time slice MT in the ST231-MT. A true simultaneous MT core supports executing multiple instructions from multiple threads in the very same cycle. Such feature would excessively impact the implementation of a VLIW microarchitecture, requiring complex capabilities like: access to any register of any threads at any cycle, full instruction dispatching to functional units to maximize performance and support for multiple independent exception handling. A time sliced MT instead supports only instructions of the same threads for a given time slice window to be dispatched down the pipeline. A time slice of 1 cycle allows threads switching every cycle thus giving better potential to reuse all threads stall cycles, while larger slices can be used to implement relatively simple priority schemes.

2.1 ST231MT architectural extensions

The ST200 architecture is a RISC VLIW embedded core family primarily aimed at multimedia and general purpose [2]. It is a 32bit, 64 general purpose register (GPR), 8 single bit branch register (BR), 4 issue machine. Elementary instructions (syllables) are packed in variable length VLIW (bundles). It comprises 4 ALUs, 2 32x32 multipliers and a single load store unit. The ST231 pipeline has 7 stages as depicted in Fig. 1.

The ST231MT is a fine-grained multithreaded machine where multiple instruction streams are selected for execution one at the time on a cycle by cycle base.

Only the F and I stages have been redesigned as the entire thread scheduling is performed in those stages. A few resources have been replicated (see Fig 1) the rest of the core being shared by all the HTs. The thread issue unit (TIU) is responsible for scheduling the threads by fetching their instructions and for issuing executable threads down the pipeline. A thread state can be one of:

- Executable: bundles can be issued from this thread.
- Stalled: the HT is waiting for a memory transaction to complete (I or D cache refill) or pipeline hazard resolution.
- Idle: the HT is waiting on an interrupt
- Waiting: the HT is waiting on given memory transaction

Every cycle, the TIU evaluates the new thread state and issues a bundle according to the scheduling policy, which can be partially controlled via software.

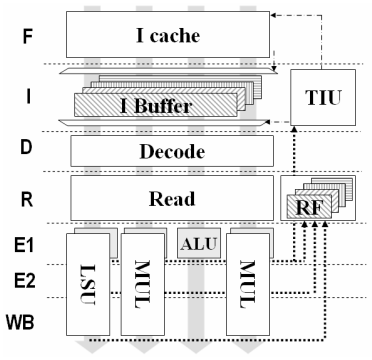

Figure 1: ST231MT pipeline

Bundles from various HTs flow down the pipe while keeping the information of the thread context they belong to for accessing its register set and private control registers, selecting correct bypass and thread private address translation logic.

In the standard ST231 core, a cache miss stalls the pipe until the cache line is filled with requested content. In the ST231MT instead, the LSU initiates a request to memory to serve the miss while the thread state is set to stalled and drained from the pipeline, freeing it for other threads ready to execute. Once the cache miss is served, the TIU is notified of this event and the stalled thread can resume execution. The LSU supports multiple outstanding misses in order to refill as many lines as hardware threads (HT) in parallel.

A specialized mechanism has been implemented to support efficient inter HTs synchronisation; to prevent threads spinning on a critical resource access and let other HTs do useful work, an HT can become idle waiting for a given memory location modification to occur. The ST231MT provides the illusion of multiple virtual CPU resulting in an increased overall core responsiveness (e.g. for almost zero cycle interrupt latency). The overall complexity increase of the ST231MT is mostly due to the large number of registers of the ST231, replicating them for each HT having an impact on the overall core size, although an optimized implementation of the register file avoids a linear increase of the size as shown on Table 1.

N° HTs	Register File		Total core	
	mm2	%	mm2	%
1	0.23	100	2.25	100
2	0.30	130	2.5	111
4	0.41	178	2.75	122
8	0.66	286	3.35	145

Table 1: Area increase for ST231 vs. ST231 MT

3. SYMMETRIC MULTIPROCESSING

The ST231MP architecture has been defined to provide system scalability and flexibility taking into account constraints of an embedded domain. Multiprocessor based systems can often result in a reduced ease of programmability because of distributed memory models limitations and to explicit communication management required at the application level. In addition, OSs can't always use efficiently hardware resources. The ST231 MP features have been defined to enable application developer to exploit peak performances by taking advantage of a cache coherent shared memory system with advantages such as: transparent tasks migration, zero la-

tency communication, interrupt distribution and with the capability to control independently processors for detached stand-alone operations, power down, etc..
Our driving criteria were: performance by minimizing latency increase due to cache coherency transactions; power consumption by reducing access to cache structures; complexity reduction and finally backward compatibility with uniprocessor solutions.
In our design, the general template for a cache coherent multiprocessor system consists of two or more ST200 cores with an L1 memory sub-system attached to a bus arbiter we call Snoop Interconnect Unit (SIU), whose tasks are coherent memory transaction management and arbitration (Fig. 2). Other components exist, such as a programmable multiprocessor interrupt controller (MPIC) devoted to interrupt distribution among the processors to provide software with interprocessor synchronization and interruption capabilities and an optional L2 cache.

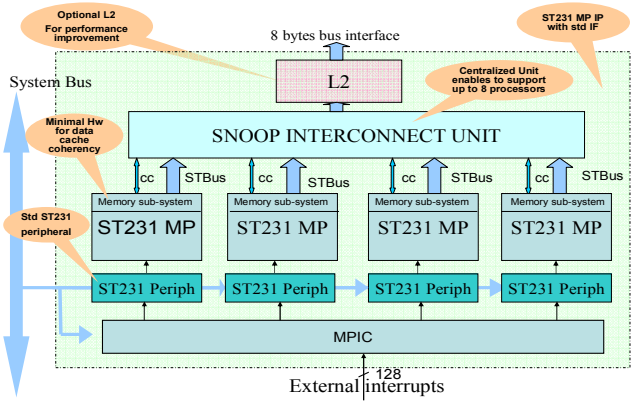

Figure 2: A 4-way smp cluster with SIU, MPIC and L2 cache

We choose a relaxed memory consistency model that simplifies the cache controller implementation and still allows for such things as intermediate buffering and transaction pipelining at a cost of a minor extension to the ISA in the form of memory barrier instructions.
Often microprocessors are implemented with a number of structures laying on the path of data between the processor core and the external memory that are used for intermediate buffering and decoupling, to offer improved performance by exploiting either temporal or spatial locality. Such structures alter the total memory operation order as seen from the external bus with respect to the issue order of the associated instructions, thus potentially affecting correctness in a cache coherent multiprocessor system. The ST231 has a write coalescing buffer of 4 entries each 32 bytes long, used to implement a hit under miss condition for write operations that miss in the cache. The introduction of a weak consistency memory model is then exposed to the programmer (often only the OS or library and not to the user level applications), by defining a memory barrier (`wmb`) instruction. This instruction is required to ensure store ordering, in fact after the execution of wmb, the processor will flush the content of the write buffer posting all of the relevant coherency transaction to the bus, making them visible from other processor instances.
For SMP architecture, shared resources access can significantly limit system performances, those accesses are mainly done through low level synchronisation routines. Efficient software synchronization requires atomic instruction support; we choose a pair of non-blocking instructions known as load linked (`ldwl`) and store linked (`stwl`). They have been selected instead of more classical (for processor of this class) test-and-set or compare–and-swap like operations because they offer lock free synchronization (e.g. for overall system responsiveness) and are better building blocks for higher level primitives. `ldwl` is a load operation which sets a lock bit associated with the load physical address. The `stwl` instead is a conditional store that checks the lock bit and, depending on its value, commits (`lock=1`) or it's "quashed" (`lock=0`), in both cases after completion the lock bit is reset. This definition allows to implement atomic primitives of various kinds (atomic increments, spin locks, etc.) with ease. To augment their flexibility we choose not to limit the type and number of instructions usable between matching pairs of `ldwl` and `stwl` as well as defining a correct behaviour independently from the associated memory region property (e.g. cached or un-cached).
We adopted a number of measures to contain the amount of additional transaction, and thus power consumption increase and performance degradation. The first consists on the ability to disconnect a processor from the coherency protocol via software, making it behave as a standard uniprocessor. The second is to reduce cache coherency overhead by using extended page attributes to activate cache coherency or not based on the application's defined memory properties. We named "coherent request" any processor request for which cache coherency is activated, "non coherent" other memory requests (data memory request or instruction refill).
Our MESI based cache coherency management is distributed between the ST231 and SIU components via a dedicated snooping interface, and not by way of a more classical bus snooping protocol. This led to an efficient solution, in terms of power and transaction latency, because coherency transactions take place only when they have to, without the need for other processors to 'listen' to every single bus transaction.

3.1 SIU/ST231MP cache coherency management

The basic principle is to split "coherent" requests from "standard" bus request. Instead of sending bus requests with coherency attributes, a processor first initiates a coherent request (opcode and address); the SIU then broadcasts it to all processors collecting a response from every one with an action associated with the original snoop request to initiate a bus operation if required. The advantages of such a scheme are minimization of broadcast latency (two cycles), and simple logic; in addition it also provides a clear decoupling with the data path used for bus arbitration.
The cache lines state (valid/dirty) management has been extended to support augmented MESI states, the data cache tags have been duplicated in order to limit contention between processor access and snooping bus access. Some of the other internal structures such as write buffer and prefetch buffer have been modified for cache coherency management, and of course extra interfaces have been implemented to manage the snooping communication protocol defined above.

3.2 Snoop Interconnect Unit (SIU)

It's a centralized component used for both bus arbitration and cache coherency management. It is partitioned in:

- Control block: manages snooping protocol interfaces, receives, serializes processor *cache coherent request*, broadcasts transaction and collects processor *snoop responses*. Those responses are used to decide arbitration among the bus request sent by processors. This arbitration scheme is used by the data path as described below.
- Data path: collects processors bus request from the standard processor bus ports and arbitrates them. Requests are either "coherent" or "non coherent", this information is associated with each processor coherency request received from the control block. Internal bus arbitration implements a fair algorithm using information coming from control path for "coherent" requests arbitration.

4. PARALLEL APPLICATION TRIALS

The applications used as benchmarks were chosen from a set of typical multimedia stream decoders, used in consumer electronic products. A video de-coder for MPEG2 [4] digital video streams and an audio decoder for Dolby AC3 [5] digital audio streams were adapted to take advantage of the SMT and SMP architectures using the threaded programming model.

The MPEG2 decoder is designed such that variable length decoding is performed by a dedicated thread and the decoded Discrete Cosine Transform (DCT) coefficients of every macro-block in a frame are stored in a memory buffer. The macro-blocks are independent units in themselves; hence multiple threads can perform the Inverse Discrete Cosine Transform (IDCT) for different macro-blocks in parallel. The Motion Compensation step exposes data dependencies that can limit the performance of the parallel implementation. Motion compensation requires data from a neighbouring macro-block in the previously decoded frame; in our implementation this data dependency is minimised by dividing the frame into two vertical halves for 2-thread implementation and into four quadrants for 4-thread implementation) and assigning the macro-blocks present in each partition to a single thread. The threads perform IDCT and motion compensation in parallel for the macro-blocks in their allotted sections. Each thread processes 1/N (N being the number of processors/threads) fraction of a picture frame. This minimizes the data dependency among processors during motion compensation since data dependencies occur only at the boundaries of the frame partitions. Simulation results show that the ratio of data dependency to the data being processed by a single processor is about 12% of the entire data processed by a thread for a 4-thread implementation. This achieves nearly perfect load balancing since all the threads are working on similar number of macro-blocks, while being mostly decoupled from each other in terms of data sets.

A packet based approach has been used to partition the Dolby AC3 audio decoder [5]. A Dolby digital audio bitstream is comprised of packets, representing equal durations of audio playback time; these packets are marshalled by headers, and are easy to located and extract from the bitstream. Packets are independent from each other and can be decoded independently; as a result, the computational load may be easily divided by assigning each thread the same number of packets to process. Packets are extracted from the bitstream by a dedicated thread, and are supplied to different computational threads, to be processed in parallel. After all threads synchronised at the end of processing, another dedicated thread arranges the decoded packets in the correct order for playback. This method achieves a high speed up with a very low data communication between threads.

5. BENCHMARKING RESULTS

In this section we first present the platforms used, the runtime environment developed to limit system overhead and finally present and discuss the results. The benchmarking goal for the MT core was to measure the obtained speedup for a range multimedia application and evaluate how MT behaves with respect to varying memory latencies. The SMP study focused on measuring timing impact of the designed cache coherent solution and evaluate achieved speedup on the proposed application.

Cycle accurate systemC and C simulation platforms were used to produce the figures illustrated in the following waiting for the actual hardware platform for final validation. A bare runtime or OS such as Linux can be mapped on such platforms.

Support of SMP platforms is becoming a common feature for various OSs; Linux has been providing such support for years now with multithreading awareness for platforms such as Intel's Hyper Threading. However Linux requires a relatively large environment and could have interfered with the benchmarking activities in complex ways; so we designed a light weight micro-kernel (a bare machine run-time) supporting I/O onto a remote host or in a local RAM file system. The newlib C library has been ported on this bare machine run-time providing common C functions. As the newlib support re-entrancy using a structured type, it has been easy to allow it to be entered by all execution contexts while each context works on his private set of data as a UNIX process would. This set up allowed us to compile and run mono threaded application unmodified by allocating each workload on a different execution context. In order to benchmark the effective speedup of applications using multiple execution contexts, we had to select a suitable parallel programming model amongst one of the many: distributed, multithread real-time or micro kernels, synchronous or streaming languages, C extensions for parallel programming or threading libraries. We have decided to use the widely adopted pthread (POSIX thread) interface as it is natively supported on most systems and does not make any assumption on the underlying machine architecture. In order to take advantage of the MT architectural features for efficient synchronisation, code has been lifted from the NPTL for the high level synchronization primitives while the lowest levels have been fully rewritten using a user level scheduler approach.

5.1 ST231MT parallel applications results

The mono threaded application core usage must be analyzed to estimate the potential speedup of an MT core. Table 2 shows the result of the execution of the parallel AC3 and MPEG2 execution compared to the same applications executing in a mono threaded fashion. Figures account for the entire execution time: micro-kernel boot, system, libc and application proper execution. In the case of parallel code, pthread synchronisation primitives represent an overhead, in the total number of instruction to be executed, especially in the case of contention. The core cycle utilisation (bundles per cycles) can provide a measure of core usage efficiency and an indication of potential for parallel execution speedup.

	Speedup	parallel code overhead %	Core cycles utilization %
MPEG2_2_MT	0.98	26	82
MPEG2_4_MT	1.06	29	91
AC3_2_MT	1.15	8	85
AC3_4_MT	1.27	8	94
MULTI-1_MT	1.09	7	64
MULTI-2_MT	1.38	9	81

Table 2: Performance scaling with varying HT numbers

The original MPEG2 code has a 67 % core utilization, leaving 33% of the cycles being unused. The parallel 2 threads version requires an extra 26% of code to be executed due to the synchronisation scheme, thus positive speedup (1.06) is only reached on 4 threads while there is almost 30% more code to be executed.

The AC3 mono threaded application shows better utilization levels as only 32% of the cycles are lost; this lowers the maximum speedup we can expect from running a parallel version of it on the MT core. For this application the need to synchronise between parallel threads is relatively low. With 4 threads we get a 1.27x speedup compared with a theoretical 1.47x with the BPC value resulting in a good 94% core utilization. We then selected two sets of multimedia applications: aac, ac3, minimad and wm9 for the first set (multi1 in table 2), and cjpeg, djpeg, mped2dec and compress for the second set (multi2). The sets are executed sequentially on the

standard ST231core while in the MT case, all applications run concurrently each on a given HT thus competing for D,/I cache and pipeline resources. As the caches have not been augmented to compensate the increased demands of all applications executing at the same time, the speedup is limited. The low utilization level for the MT case is due to both capacity and collision misses in the caches.

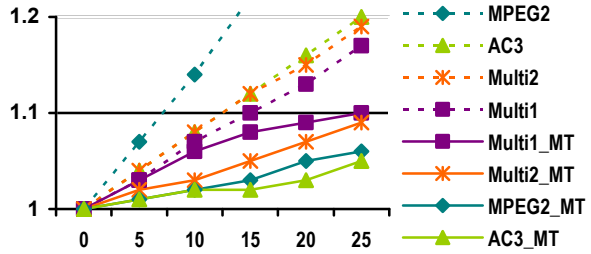

Figure 3: Slow down as a function of memory latency

Figure 3 shows the effect on the overall execution time of varying memory latencies (number of system bus cycles for servicing a memory request). For all the applications, the MT executions show a greatly reduced impact as opposed to the sequential execution.

5.2 Cache coherency overhead

To evaluate cache coherency overhead we have used three different platforms: a first one uses just one processor without cache coherency used as the baseline, the second still with one core but with cache coherency active useful to assess latency increase, and finally a third platform integrates four processors, running the same application, but different executable images and set of data thus without explicit data sharing except for the I/O libraries state variables.

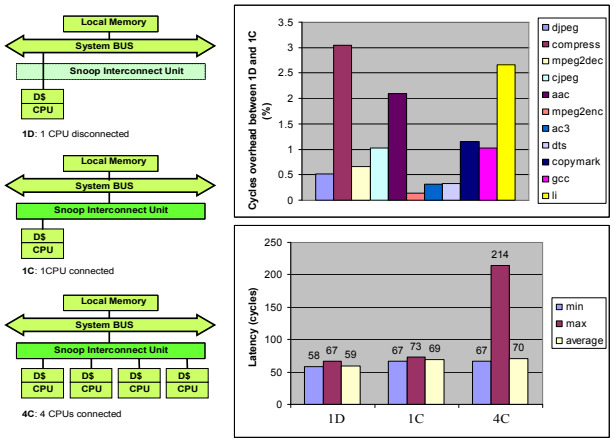

Figure 4: cache coherency performance impact

Results show that the cycle overhead is between 0.2 and 3% for a single processor with cache coherency activated. These figures can be explained with the load miss penalty increase of up to 9 cycles due to cache coherency and the percentage of read misses versus number of total memory operations. The overhead gets up to 3.7% when four processors are running but is mainly due to bus contention because of the four independent applications accessing the bus and not significantly impacted by cache coherency.
The second diagram in Fig. 4 shows load miss latency; when four processors are running, the maximum latency for a load miss is around 4 times larger than the uniprocessor case. This can be explained by a side effect of the I/O library critical section contention.

5.3 ST231 SMP Parallel application results

The MPEG2 and AC3 parallel applications described in previous chapters have been executed on the cycle accurate System C platform with the support of the same runtime of the MT environment. We expected to find, for the AC3 application, a significant speed up with respect to the MT results. Indeed the parallel code fully exploits the multiple processor resources; Table 3 shows linear speed ups with the number of processors, combining MT and SMP execution would provide beneficial cumulative effect by also improving single core utilization.

	1 core	2 cores	3 cores	4 cores
AC3	1	1.8	2.9	3.8
MPEG2	1	1.1		1.5

Table 3: AC3 and MPEG2 parallel applications speed up

For MPEG2 trials, the results also differ from the MT case and point out the need for a more careful partitioning of workload amongst parallel tasks. In fact the sequential nature of the variable length decoder limits the potential speedup to 1.3 with two processors, as a function of the bit rate and resolution used for the decoded stream. This preliminary results are inconclusive for the MPEG2 parallel implementation as a different parallelization strategy would probably provide better results.

6. CONCLUSIONS

We've demonstrated that two orthogonal microprocessor architectural features, SMT and SMP support, can have a significant impact on the performance of multimedia streaming application. To our knowledge these are the first reported implementations of simultaneous multithreading and cache coherency in the context of an embedded VLIW general purpose core with precise exception. Results show that interesting speed ups can be obtained at a relatively small extra hardware cost for both independent multiple workloads and parallel implementation of media applications. However it's also evident from the preliminary results of our implementation of a video decoder that a suitable programming model and a careful parallelisation strategy need to be applied to obtain good results in the presence of a balanced mix of control and data flow.
Future work will focus on the design of more advanced support for streaming in the form of distributed memory architectures and to the definition of programming model and hardware support that removes some of the difficulties of application partitioning on multiple cores.

7. REFERENCES

[1] http://www.cs.clemson.edu/~mark/multithreading.html

[2] P. **Faraboschi**, G. **Brown**, J. A. **Fisher**, G. **Desoli**, F. **Homewood**: *Lx: a Technology Platform for Customizable VLIW Embedded Processing* 27th Annual International Symposium on Computer Architecture -- ISCA'00, June 2000.

[3] A. **Moshovos**, "*RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence*", Intl. Symp. on Computer Architecture, pp. 234-245, 2005.

[4] ISO/IEC 13818-2: "*MPEG video standard*", ITU-T H.262 Recommandation, 1995

[5] *Digital Audio Compression Standard* AC-3, Document no. A/52, Advanced Television Systems Committee, 1995