ISCAS 2000 - IEEE International Symposium on Circuits and Systems, May 28-31, 2000, Geneva, Switzerland

PERFORMANCE EVALUATION OF MULTITHREADED ARCHITECTURES FOR MEDIA PROCESSING APPLICATIONS

S. Balakrishnan and S. K. Nandy

Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore 560 012, India.
{sbalki, nandy}@serc.iisc.ernet.in

ABSTRACT

In this paper, we present an architecture framework called *SYMPHONY* consisting of a linear array of processors that exploits parallelism in media applications at micro (ILP) and macro (threads) levels. High speed communication and synchronization necessary for efficient multithreading is achieved using novel hardware and software mechanisms. We demonstrate the efficacy of the SYMPHONY framework by performance evaluation of an instance in the framework. We show that multithreaded architectures coupled with SIMD parallelism provides performance improvement in excess of 2X over conventional superscalar architectures.

1. INTRODUCTION

Advances in deep submicron technologies have made single chip systems viable [1]. These system integrate programmable processor cores and dedicated hardware components onto a chip. A high-end multimedia processing chip, for example, may integrate programmable RISC/DSP processor cores along with special hardware accelerators, functional units and memory to achieve high performance. These *systems-on-a-chip* (SoC) often use multiprocessor architectures to meet the performance demands of applications that are multi-functional by nature [2].The design of SoC is extremely complex because early design decisions tend to reflect on the performance of the system and also because chip design is driven by rapid time-to-market needs. With new standards being defined for various high performance applications [3, 4, 5], it is advantageous to provide programmability, reconfigurability and scalability in such SoCs, wherein dramatic improvements in speed can be achieved by fine tuning a few parameters in the architecture to suit specific applications. With these constraints in mind, it becomes easier for system designers to base their chips on a broad architecture framework that provides solutions for a domain of applications and then tune the reconfigurable elements in the architecture to suit the application in question. The SYMPHONY architecture framework [6] seeks to achieve these goals for media applications.

Media applications exhibit a lot of inherent parallelism both at the micro level [7] (Instruction Level Parallelism) and at the macro level [8] (threads). While ILP can be exploited using techniques similar to that used in conventional superscalar processors [7], the extent to which thread level parallelism can be exploited is predicated by the application of fast communication and synchronization techniques. Dramatic performance benefits in media applications can be achieved by exploiting single instruction multiple data (SIMD) parallelism. To speedup media applications, commercial processors have implemented SIMD style arithmetic and instructions have been provided to exploit SIMD parallelism [9]. Here instruction operate on sub-datatypes of a 64 bit register in parallel. The SYMPHONY framework exploits all the kinds of parallelism detailed above.

In this work we have architected a set of development tools for a system in the SYMPHONY framework consisting of simultaneously multithreaded superscalar processors. We use an execution-driven processor simulator to evaluate the architecture framework for specific configurations of the SYMPHONY instance. Each thread in a configuration under consideration consists of a small issue width superscalar processing unit. Execution times of benchmark programs are compared against that of a wide issue unithreaded superscalar architecture.

The rest of the paper is organized as follows. In Section 2 we discuss the SYMPHONY framework for media processing. In section 3 we describe the experimental setup and discuss results. We conclude in section 4.

2. THE SYMPHONY ARCHITECTURE FRAMEWORK

Having motivated the need for a framework based design for media processors, in this section we present the SYMPHONY framework. A system in the SYMPHONY framework consists of a linear array of SYMPHONY processors (SPs) as shown in Fig. 1. We use bidirectional busses for interconnecting processors. Each processor has to its own private memory connected via private instruction and data caches. Other system interfaces for streaming data and display buffers that are part of a any multimedia system have not been included in the figure.

A processor in the SYMPHONY framework (SP) has hardware to support simultaneous multithreading [14, 15, 13]. A broad organization of an SP is shown in Fig. 1. An SP has a few extra hardware modules not seen in multithreaded architectures proposed earlier. These are the communication registers, the single assignment memory (SAM) module and the communication controller (CC). In addition to these modules we also have integer functional units and floating point units along with special media processing functional units that can perform arbitrary width SIMD style arithmetic [11]. SIMD style processing is similar to that used in general purpose processors with media extensions (*eg.* MMX [10]).

Each SP, in addition to the general purpose registers, also has registers called *transfer registers* that are used for communicating fine grained data between processors. Register variables that a program segment or task is guaranteed to produce and later consumed

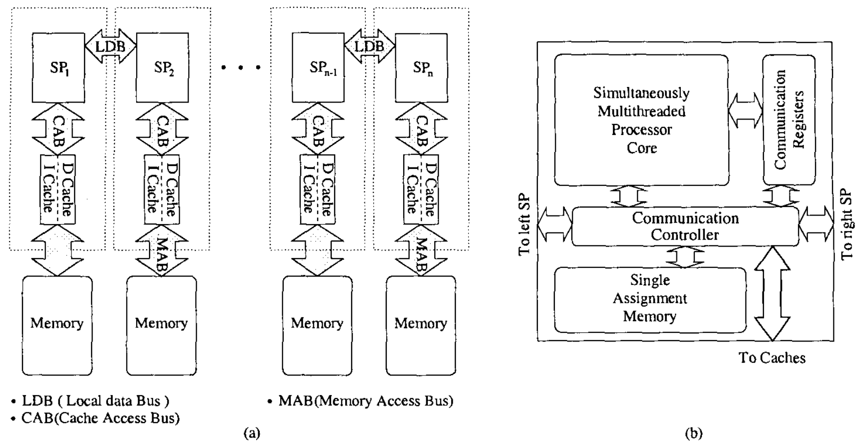


Figure 1: SYMPHONY: (a) System Organization (b) Modules in an SP

by another task executing in a neighboring processor may be assigned a transfer registers. The use of the transfer registers must be determined statically so that the consumer task can know *a-priori*, the registers in which it can receive data. We advocate the use of efficient compiler techniques to overlap communication with computation to a large extent. The transfer register file hence, consists of a transfer register set T and a shadow register set SH that is not visible to the program. During program execution the current set of visible registers T is used while the SH registers are written into with values communicated from neighboring SPs. Multiple writes to the same registers is buffered by the CC. The CC can be programmed to partition the SH set of registers into subsets T_{left} and T_{right} corresponding to values communicated from the left and right neighboring SPs respectively. An explicit `switch` instruction has to be executed when the values that have been written into the SH registers are to be used. On executing a `switch` instruction the T and SH registers are swapped making the values in the SH registers visible to the program. To guarantee that a switch instruction does not lose data meant for a subsequent task, the program sets a mask using a `setmask` instruction, indicating the registers in which it expects values for the next task to be scheduled onto the SP. A write into one such register resets the mask bit associated with it. A switch instruction executed subsequently will block until all the registers indicated by the mask have been updated.

In general purpose processing the data cache buffers part of the main shared address space. Data caches are very effective in accessing data that have temporal locality and whose access patterns are non-predictable. The access times for data is hence non-predictable and on a cache miss can be an order of magnitude longer than on a cache hit. This however, is undesirable for media processing applications that have real-time constraints. We therefore use a special memory module called the single assignment memory (SAM) that has access times comparable to that of a cache hit. Shared variables that have predictable access patterns or whose lifetimes are predictable are targeted to SAM. It should be noted that SAM is effective for data that are small to medium sized since it is used for synchronization between threads and for data with predictable locality. The use of SAM is similar in spirit and concept to static single assignment — used extensively in compilers for data flow analysis. A bit associated with every word in SAM indicates the presence or absence of data. A hardware scheme implemented in the CC can unblock threads waiting for data whenever the requested data arrives.

The CC coordinates the use of the communication registers, SAM and accesses to the cache. The SAM modules of all the SPs, together give the appearance of a single shared memory module. Each SP, on startup, is configured using special supervisory instructions to use disjoint segments of this shared data space. To direct accesses to SAM that belongs to another processor, a CC maintains the address mappings of the processors in the system. Memory operations in a particular processor can thus be directed to the SAM of the same processor, one of the neighboring processors or to a processor's local memory. The CC distinguishes these accesses and directs them either to the appropriate processor, local SAM or the cache controller.

3. EXPERIMENTAL SETUP AND RESULTS

In this work we have built a set of development tools for a system in the SYMPHONY framework consisting of simultaneously multithreaded superscalar processors. Simulation tools to verify and evaluate the architecture framework for specific configurations of the SYMPHONY instance have also been built. Each thread in the configuration under consideration consists of a small issue width superscalar processing unit. We have defined a multithreaded extension to the instruction set architecture of the MIPS processor for our simulation studies. The aim of our study was to explore the use of superscalar processor cores like the R10000 to do simultaneous multithreading. In this paper we report preliminary studies on the use of a multithreaded uniprocessor instance of the SYMPHONY framework for media applications.

A high performance simultaneously multithreaded media processor would require fine-grained communication and synchronization between threads. We use the single assignment memory described earlier for this purpose. The SMT processor proposed by Tullsen *et al* [14] shares the instruction scheduling unit among all executing threads. The architecture would deadlock if a blocked thread fills up the instruction queue preventing a release instruction from another thread from executing in the processor. A solution in which instructions from the blocked thread are flushed from the instruction queue has been proposed in [16]. We however, use separate instruction queue structures for each thread, thereby alleviating the problem.

Tools in this environment consist of a compiler and a simulator. The compiler for the specific instance of SYMPHONY mentioned above takes annotations specifying threads and shared synchronization variables and assigns them to registers or single assignment memory. The compiler produces object code for a multithreaded extension of the MIPS architecture. Extensions to the MIPS architecture are in the form of instructions to setup new thread contexts, spawn new threads and media instructions to handle arbitrary precision arithmetic. The execution-driven processor simulator mimics the processor pipeline of a MIPS R10000 out-of-order superscalar processor [12] for each thread. A branch prediction buffer shared among all threads is also used for branch prediction. Considering the streaming nature of media applications we include only a single level of cache. We use the ICOUNT.2.8 fetch policy described in [14] to use the fetch bandwidth effectively.

3.1. Workloads and results

Media processing applications are characterized by tight "inner-loops" that execute repeated computations on different set of data. These loops are amenable to parallelization. Parallelism in these loops can be exploited through ILP, SIMD processing and multithreading. Further, these loops can be classified into: those that have data dependent flow of control and those with static flow of control. MPEG-2 encoding involves motion estimation and falls into the first class wherein more amount of processing has to done on scenes involving higher activity (motion). Adaptive filtering [17] on the other hand involves loops that have a highly deterministic flow of control[1]. In this work we study two such benchmarks for various configurations of the SYMPHONY instance. We compare the performance of various multithreaded architecture configurations against configurations of conventional superscalar processors. The base processor and memory configurations are tabulated in tables 1 and 2 respectively.

One of the benchmarks that has been chosen is the MPEG-2 encoder of the MPEG Software Simulation Group (MSSG). The encoder was run with 160×120 pixel 3-band images with high activity (rocket.mpg — A rocket launching scene). The rocket scene consists of 50 frames. The scenes were encoded at a frame rate of 25fps, bit rate of 5Mbps at Main Profile @ Main Level (MP@ML). The images were in 4:2:0 YUV chroma format. Default quantization tables of the MSSG are adopted. Another benchmark that was chosen is the digital adaptive equalizer for broadband modems. The adaptive equalizer consists of a 120-tap delayed least mean square (DLMS) adaptive filter [17]. The tight loops of the DLMS filter exercise the SAM and the performance figures indicate the efficacy of SAM as a mechanism that supports low overhead fine-grained synchronization.

Figures 2 – 3 compare the performance of various architectures relative to a baseline architecture indicated in each figure. We represent the configuration of an architecture with a tuple as described below:

$$\langle sc, iw, ac, ia, im, sa, ss \rangle$$

[1]Adaptive filtering constitutes the heart of the computations in adaptive equalizers for broadband modems

Branch Prediction	
Bimodal predictor size	1K
Taken Branches per cycle	1
Simultaneous speculated branches	5
Maximum No. of thread contexts	5
Functional unit latencies	latency/repetition rates
Default Integer	1/1
Integer multiply	3/1
Integer divide	11/10
Default floating point	3/1
floating moves/converts	2/2
floating point divides	14/14
Default SIMD	1/1
Default SIMD multiply	3/1
Integer loads	2/1
floating point/SIMD loads	3/1

Table 1: Default processor parameters

Cache line size	64 bytes
Data Cache size	16 KB
Data cache request ports	2
Cache hit time	2 cpu cycles
Cache read miss penalty	50 cpu cycles
Cache write miss penalty	15 cpu cycles

Table 2: Default memory system parameters.

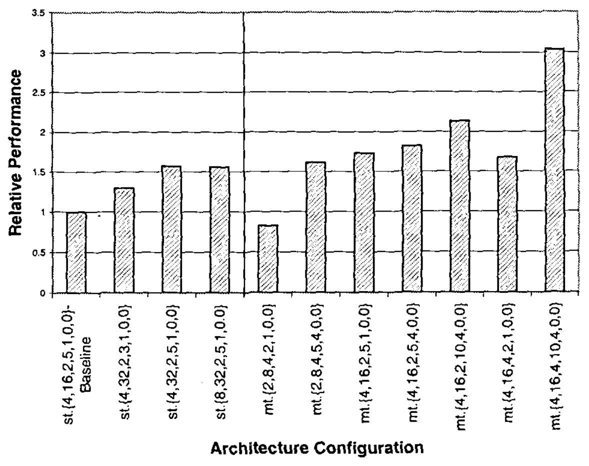
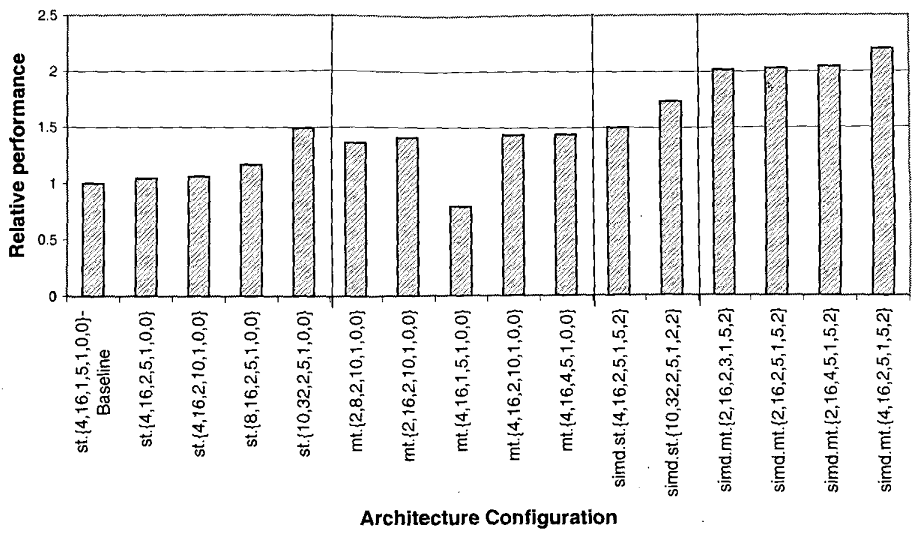


Fig 2. Relative performance of various configurations for the DLMS kernel

where *sc, iw, ac, ia, im, sa, ss* indicate the scalarity, instruction window size, associativity of the data cache, number of integer ALUs, number of integer multipliers, number of SIMD ALUs and the number of SIMD shift units respectively. Prefixes to the tuple could be a combination of *simd* with *st* or *mt*, where, *simd* indicates that SIMD parallelism is exploited by the architecture instance. *st* or *mt* indicate that the instance is unithreaded or multithreaded. The following inferences can be drawn from the figures:

- In the MPEG-2 encoder, performance of superscalar processors improve marginally with increase in the number of functional units for window sizes less than 16. Doubling the

Fig 3. Relative performance of various configurations for the MPEG2 encoder benchmark (rocket.mpg)



size of the instruction windows gives a performance boost of as much as 50%. This indicates that ILP is far flung in such applications.

- Multithreading provides comparable performance to a wide issue superscalar processor without the attendant penalty of a slower clock (not accounted for in performance figures). Since there is only one level of cache, an efficient cache organization is necessary. It is seen that the increase in associativity prevents cache thrashing due to accesses from multiple threads.
- SIMD parallelism gives significant performance gains in both the benchmarks. This is because of exploitation of data parallelism and also the reduction of control flow overheads.
- Performance can be enhanced by a further 50% if an architecture is able to exploit both SIMD and multithreading. This however, comes at the cost of more SIMD ALUs.
- The DLMS kernel benefits from the use of more functional units. This is due to higher levels of data parallelism.

4. CONCLUSIONS

In this paper, we have presented an architecture framework called *SYMPHONY* that exploits parallelism in media applications at micro (ILP) and macro (threads) levels. We have demonstrated the efficacy of the SYMPHONY framework by performance evaluation of an instance in the framework. We have shown that multithreaded architectures coupled with SIMD parallelism provides performance improvement in excess of 2X over conventional superscalar architectures.

5. REFERENCES

[1] John Hennessy. The Future of Systems Research. In *IEEE Computer Magazine*, pp 27–33, Vol 32, No. 8, August 1999.

[2] S. Santhanam *et al.*. A Low-cost, 300MHz, RISC CPU with Attached Media Processor. In *IEEE Jl. of Solid-State Circuits*, pp 1829–1839, Vol 33, No. 11, November 1998.

[3] MPEG-4 Standard from ISO, http://www.mpeg.org/MPEG

[4] Very high-rate Digital Subscriber Line, http://www.vdsl.org

[5] UMTS, http://www.umts-forum.org

[6] S. K. Nandy, S. Balakrishnan, Ed. Deprettere. SYMPHONY: A Scalable High Performance Architecture Framework for Media Applications. In *Proc. of the Fifth Intl. Conf. on Advanced Computing, Chennai, India*, Dec. 1997.

[7] James E. Smith and Gurinder S. Sohi. The Microarchitecture of Superscalar Processors. In *Proc. of the IEEE*, pp 1609–1624, Vol 83, No. 12, December 1995.

[8] Jack L. Lo *et al.*, Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading. In *The ACM Transactions on Computer Systems*, 1997.

[9] P. Ranganathan *et al.* Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions. In *Proc. 26th Intl. Symp. Computer Arch.*, 1999.

[10] Alex Peleg *et al.*, Intel MMX for Multimedia PCs. In *Communications of the ACM*, pp 25–38 Vol. 40, No. 1, January 1997,

[11] S. Balakrishnan and S. K. Nandy. Arbitrary Precision Arithmetic-SIMD Style. In *Proc. of the 11th Intl. Conf. on VLSI Design, Chennai, India*, January 1998.

[12] Kenneth C. Yeager. The MIPS R10000 Superscalar Microprocessor. In *IEEE Micro*, April 1996.

[13] G. S. Sohi *et al.*, Multiscalar Processors. In *Proc. 22nd Intl. Symp. on Computer Arch.*, pp 414–425, June 1995.

[14] Dean M. Tullsen *et al.*, Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, In *Proc. 23rd International Symposium on Computer Architecture*, 1996.

[15] J-Y. Tsai and P-C. Yew. The Superthreaded Architecture: Thread Pipelining with Run Time data Dependence Checking and Control Speculation. In *Intl. Conf. on Parallel Arch. and Compilation techniques*, pp 35–46, Oct. 1996.

[16] Dean M Tullsen *et al.* Supporting Fine-Grained Synchronization on a Simultaneous Multithreading Processor. *Proc. of the 5th Intl. Symp. on High Perf. Comp. Arch.*, Jan. 1999.

[17] Chris J. Nicol *et al.* A Low Power 128-Tap Digital Adaptive Equalizer for Broadband Modems. In *IEEE Journal of Solid-State Circuits*, pp 1777–1789, Vol. 32, No. 11, November 1997.