Asian Journal of Information Technology 5 (10): 1137-1141, 2006

A Parallel Processing Approach to Image Processing Application Using Simultaneous Multi Threading

K. Manjunathachari
Department of ECE, G. Pulla Reddy Engineering Collge, Kurnool-518004, A.P, India

Abstract: Typical real time image processing applications require a huge amount of processing power, computing ability and large resources to perform the image processing applications. The limitations appear on image processing systems due to the volumetric data of image to be processed. This challenge is more dominant when coming to process the image processing applications parallely. Parallel processing appears to be the only solution to attain higher speed of operation at real time resource constraints. The nature of processing in typical image processing algorithms ranges from large arithmetic operations to fewer one. Although the existing parallel computing systems provide to some extent parallelism for image processing but fails to support image processing operations varying at a large rate. As part of my thesis, this study presents a novel parallel processing approach to image processing applications using simultaneous multithreading by bifurcating the operations of parallelism into three distinct layers.

Key words: Bucket processing, SIMD, MIMD, simultaneous multithreading

INTRODUCTION

The type of processing operations in a typical image processing task varies greatly. Generally three levels of image processing are distinguished to analyze and tackle the image processing application: low-level operations, intermediate-level operations and high-level operations.

Low-level operations: Images are transformed into modified images. These operations work on whole image structures and yield an image, a vector, or a single value. The computations have a local nature; they work on single pixels in an image. Examples of low-level operations are: smoothing, convolution, histogram generation.

Intermediate-level operations: Images are transformed into other data structures. These operations work on images and produce more compact data structures (e.g., a list). The computations usually do not work on a whole image but only on objects/segments (so called areas of interest) in the image. Examples of intermediate-level operations are: region labeling, motion analysis.

High-level operations: Information derived from images is transformed into results or actions. These operations work on data structures (e.g., a list) and lead to decisions in the application. So high-level operations can be characterized as symbolic processing. An example of a high-level operation is object recognition.

A image processing starts with a plain image, or sequence of images, (coming from a sensor) and, while processing, the type of operations moves from arithmetic (Floating Point Operations Per Second, FLOPS) to symbolic (Million Logic Inferences Per Second, MLIPS) and the amount of data to process is reduced until in the end some decision is made (image understanding). As may be obvious, image processing tasks require large amounts of (different type of) computations. When real-time requirements are to be met, normal (sequential) workstations are not fast enough. So more processing power is needed and parallel processing seems to be seems to be an economical way to satisfy these real time requirements. Besides even when current workstations get fast enough to do the image processing task of today, parallel processing will offer more processing power and open new application areas to explore.

Many architectures have been proposed that try to exploit the available parallelism at different granularities. For example, pipelined processors[1-3] and multiple instruction issuing processors, such as the superscalar[4-6] and VLIW[4-5] machines, exploit the fine-grain parallelism available at the instruction set level. In contrast, shared memory multiprocessors[6,7] exploit coarse-grain parallelism by distributing entire loop iterations to different processors. Each of these parallel architectures have significant differences in synchronization overhead, instruction scheduling constraints, memory latencies and implementation details, making it difficult to determine which architecture is best able to exploit the available

parallelism. The performance potential of multiple instruction issuing and its interaction with pipelining has been investigated by several researchers[8-10]. Their work has shown that at the basic block level, pipelining and multiple instruction issuing are essentially equivalent in exploiting fine-grain parallelism. Studies using the PASM prototype have indicated that the multiprocessor organization may be outperformed by the SIMD organization[4,12] unless special care is taken to provide efficient synchronization for the MIMD mode[13]. We extend this previous work by comparing the performance of a pipelined processor, a superscalar processor and a shared memory multiprocessor when executing scientific application programs.

In image processing applications the existing approach to parallelism get constrained due to variant size of data and the required resources. Hence a system is required for the efficient controlling of image processing application with variable data size. The proposed approach realizes a parallel processing architecture integrating the Simultaneous Multithreading Concept (SMT) for the proper control and execution of variant image processing application.

MULTITHREADING AND SMT

Simultaneous multithreading is a processor design that combines hardware multithreading with superscalar processor technology to allow multiple threads to issue instructions each cycle. Unlike other hardware multithreaded architectures (such as the Tera MTA), in which only a single hardware context (i.e., thread) is active on any given cycle, SMT permits all thread contexts to simultaneously compete for and share processor resources. Unlike conventional superscalar processors, which suffer from a lack of per-thread instruction-level parallelism, simultaneous multithreading uses multiple threads to compensate for low single-thread ILP. The performance consequence is significantly higher instruction throughput and program speedups on a variety of workloads that include commercial databases, web servers and scientific applications in both multiprogrammed and parallel environments.

Simultaneous multithreading has already had impact in both the academic and commercial communities. The project has produced numerous papers, most of which have been published in journals or the top, journal-quality architecture conferences and one of which was the most recent paper selected for the 25th Anniversary Anthology of the International Symposium on Computer Architecture, a competition in which the criteria for acceptance was impact. The SMT project at the University of Washington has also spawned other university projects in simultaneous multithreading. Lastly, several U.S. chip manufacturers (Intel, IBM, Sun and Compaq (when it still supported the Alpha microprocessor line) have designed and manufactured SMT processors for the high-end desktop market. Several startups are also building SMT processors

Conventional processors execute instructions from a single instruction stream. Despite micro architectural advances, execution unit utilization remains low in today's microprocessors. It is not unusual to see average execution unit utilization rates of approximately 25% across a broad spectrum of environments. To increase execution unit utilization, designers use thread-level parallelism, in which the physical processor core executes instructions from more than one instruction stream. To the operating system, the physical processor core appears as if it is a symmetric multiprocessor containing two logical processors. There are at least three different methods for handling multiple threads. In coarse-grained multithreading, only one thread executes at any instance.

When a thread encounters a long-latency event, such as a cache miss, the hardware swaps in a second thread to use the machine's resources, rather than letting the machine remain idle. By allowing other work to use what otherwise would be idle cycles, this scheme increases overall system throughput. To conserve resources, both threads share many system resources, such as architectural registers. Hence, swapping program control from one thread to another requires several cycles. IBM implemented coarse-grained multithreading in the IBM eServer pSeries Model 680.2 A variant of coarse-grained multithreading is fine-grained multithreading. Machines of this class execute threads in successive cycles, in round-robin fashion 3. Accommodating this design requires duplicate hardware facilities. When a thread encounters a long-latency event, its cycles remain unused. Finally, in simultaneous multithreading (SMT), as in other multithreaded implementations, the processor fetches instructions from more than one thread.4 what differentiates this implementation is its ability to schedule instructions for execution from all threads concurrently. With SMT, the system dynamically adjusts to the environment, allowing instructions to execute from each thread if possible and allowing instructions from one thread to utilize all the execution units if the other thread encounters a longlatency event.

Enhanced SMT features: To improve SMT performance for various workload mixes and provide robust quality of service, we added two features to the Power5 chip: dynamic resource balancing and adjustable thread priority.

Dynamic resource balancing: The objective of dynamic resource balancing is to ensure that the two threads executing on the same processor flow smoothly through the system.

Dynamic resource-balancing logic monitors resources such as the GCT and the load miss queue to determine if one thread is hogging resources. For example, if one thread encounters multiple L2 cache load misses, dependent instructions can back up in the issue queues, preventing additional groups from dispatching and slowing down the other thread. To prevent this, resource-balancing logic detects that a thread has reached a threshold of L2 cache misses and throttles that thread. The other thread can then flow through the machine without encountering congestion from the stalled thread. The Power5 resourcebalancing logic also monitors how many GCT entries each thread is using. If one thread starts to use too many GCT entries, the resource balancing logic throttles it back to prevent its blocking the other thread.Depending on the situation, the Power5 resource-balancing logic has three threadthrottling mechanisms:

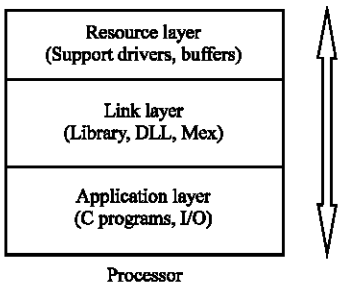
- Reducing the thread's priority is the primary mechanism in situations where a thread uses more than a predetermined number of GCT entries.
- Inhibiting the thread's instruction decoding until the congestion clears is the primary mechanism for throttling a thread that incurs a prescribed number of L2 cache misses.
- Flushing all the thread's instructions thatare waiting for dispatch and holding the thread's decoding until the congestion clears is the primary mechanism for throttling.

The SMT model: This section briefly describes the SMT architecture and our SMT simulator. In general, we use the term thread to refer to a schedulable sequential task (an independent program or a component of a parallel program) and the word context to refer to the hardware structures that hold an executing thread's processor state. At any instant, a context holds a thread and there are more threads than contexts.

An SMT processor executes multiple instructions from multiple threads each cycle. Even with this increased functionality, SMT can be constructed with straightforward modifications to a standard dynamically-scheduled superscalar processor[19]. To support multiple resident threads, several structures must be replicated or modified. These include per-context thread state (registers and a program counter), active lists and mechanisms for pipeline flushing, traps, interrupts and return stack prediction. In addition, the branch-target buffer and TLB entries must include thread IDs. Modifications of this nature increased the chip area of Compaq's recently announced Alpha implementation of SMT by only 10%, compared to a similar superscalar design[15].

PARALLEL PROCESSING APPROACH TO IMAGE PROCESSING APPLICATION

In this study we present a method to the bifurcation of image processing application into three fundamental layers which are isolated based on processor requirements and their functionality. Generally the parallel computing image processing applications perform parallel operations by taking additional resource support from library and packages and create buffering for performing image PA. The transition of control for creating buffers and controlling the applications takes a considerable amount of transfer time which results in slower processing. We present an approach to enhance the parallelism by adding the concept of simultaneous multithreading over the processor for redundancy the transition delay in Parallel Computing Image Processing Application. The Parallel Computer Architecture is layered into three regions as shown below.



Resource layer: This layer provides the track of all the hardware resource requirements such as device drivers, processing buffers for performing multiple IPA. This layer communicates with the application layer via linking layer to find the requirements of IP applications so as to allocate processing buffers to carry simultaneous operations.

Linking layer: This layer provides a link between the resource layer and application layer which consists of DLL files and mex files for proper transfer of data between

resource allocation unit and computing unit. This layer hold the library defined and the packages required for supporting the transactions.

Application layer: This layer reads the input image and dedicated functions. IPA with support of upper layer. This layer evaluates the time of computation and the resource requirement for IPA. A copy of requirements is transferred to resource layer for allocation of resources. This layer is the User I/F where the user can pass the inputs to be processed on the image and obtain results. The transactions in these layers are controlled by the simultaneous multi threading approach where the instructions are latched out into multiple threads and are executed concurrently. Finally, in Simultaneous Multithreading (SMT), as in other multithreaded implementations, the processor fetches instructions from more than one thread 4. What differentiates this implementation is its ability to schedule instructions for execution from all threads concurrently. With SMT, the system dynamically adjusts to the environment, allowing instructions to execute from each thread if possible and allowing instructions from one thread to utilize all the execution units if the other thread encounters a long latency event.

RESULTS AND DISCUSSION

The above is technique is simulated and the results are tested on a P-III, 1.72GHz with 256MB RAM

Standard test Images that are considered: flower, TUD, obscura, trui, cermet.

We shall now examine how SMT performs on standard test image flower and compare its performance with the test image on Fine Grain Multithreading (FGM) and Coarse Grain Multithreading (CGM). From the above Fig. 1 as the number of threads increases by keeping the image size fixed it shows that the execution time of SMT based low pass filtering is better then FGM based low pass filtering and CGM based low pass filtering.

Elements in the processing buffer at the start of each iteration for image processing iterations of the test images

Image name	Entries in processing buffers
Flower	1500
TUD	3800
Obscura	200
Trui	9000
Cermet	10000

Image size	SMT(ms)	CGM(ms)	FGM(ms)
64*64	20	25	28
256*256	200	220	250
400*400	220	245	260

Comparison of Execution time on various image sizes

The analysis for the following are under development:

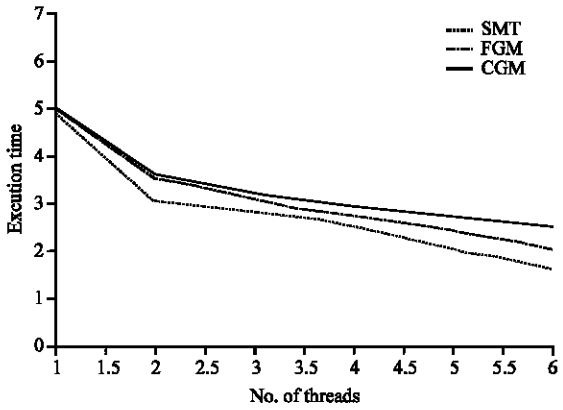


Fig. 1: Execution time of SMT based on FGM and CGM

- Tradeoff for image processing operations using different number of processors.
- Speedup of SMT versus Local Neighborhood Operations, Region Neighbor Operations
- Execution for varios Image processing operations using different number of threds.

REFERENCES

1. Anderson, D.W. *et al.*, 1967. The BM System/36O Model 91: Machine philosophy and instruction-handling. IBM J. Res. d Deu.
2. Jouppi, N.P., 1989. Architectural and organizational tradeoffs in the design of the multititan CPU. ISCA, pp: 281-289.
3. Radin, G., 1983. The 801 Minicomputer. IBM J. Res. 8 Dev., pp: 237-246.
4. Fisher, J.A., 1981. Trace scheduling: A Technique for global microcode compaction. IEEE TOC, pp: 478-490.
5. Lam, M., 1988. Software pipelining: An effective scheduling technique for VLIW machines. SIGPLAN '88, pp: 318-328.
6. Gottlieb, A. *et al.*, 1982. The NYU Ultracomputer-designing a MIMD, Shared-Memory Parallel Machine. ISCA, pp: 27-42.
7. Pfister, G.F. *et al.*, 1985. The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture. ICPP, pp: 764-771.
8. Pleszkun, A.R. and G.S. Sohi, 1988. The performance potential of multiple functional unit processors. ISCA, 1988, pp: 37-44.
9. Smith, M.D. *et al.*, 1989. Limits on multiple instruction issue. ASPLOS, Apr. 1989, pp: 290-302.
10. Sohi, G.S. and S. Vajapeyam, 1989. Tradeoffs in instruction format design for horizontal architectures. ASPLOS, pp: 15-25.

11. Fineberg, S.A. *et al*., 1987. Mixed-mode computing with the PASM System Prototype. Allerton Conf. Comm., Con. Comp., pp:
12. Fineberg, S.A. *et al*., 1988. Non-Deterministic Instruction Time Experiments on the PASM System Prototype. ICPP, pp: 444-451.
13. Bronson, E.C. *et al*., Experimental Application-Driven Architecture Analysis of an SIMD/MIMD Parallel Processing System.
14. Kunkel, S.R. and J.E. Smith, 1986. Optimal pipelining in supercomputers. ISCA, pp: 404-411.
15. Diefendorff, K., 1999. Compaq chooses SMT for alpha. Microprocessor Report, pp: 13-6.