Evaluation of Various Node Configurations for Fine-grain Multithreading on Stock Processors*

Jin-Soo Kim Soonhoi Ha Chu Shik Jhon

Seoul National University
Department of Computer Engineering
Seoul 151-742, KOREA
{jinsoo, sha, csjhon}@comp.snu.ac.kr

Abstract

It becomes more and more interesting to construct multithreaded parallel machines using stock processors due to their high performance/price ratio. However, no quantitative analysis has been reported on the effectiveness of various node configurations and its impact on the overall performance. In this paper, we explore three different node configurations in detail and compare their dynamic characteristics through the instruction-level simulation with six benchmark programs. Our experiments show that employing a dedicated processor for communication and synchronization is a reasonable approach because it can almost double the performance. Several factors that limit the overall speedup are also presented.

1. Introduction

Large-scale parallel computers are expected to have communication latencies of several tens to hundreds of processor cycles. The latency becomes a problem if the processor spends a large fraction of computation time sitting idle waiting for remote operations to complete. Multithreading has been proposed as a basic mechanism for future parallel systems due to its inherent ability to tolerate communication latency. Multithreading allows a processor to hide latency by switching from one thread to another when a long latency operation is encountered.

Traditionally, multithreaded architecture efforts tried to redesign substantial part of the processing unit in order to take full advantage of the multithreaded execution model and to minimize overheads. However, microprocessor performance is advancing at a rate of 50 to 100% per year [3] and this trend still appears to be possible for the next few years. Because microprocessors are produced in very high volumes, their development costs can be easily amortized. Better performance/price ratio of such microprocessors already led Intel, Thinking Machines, Meiko, Convex, IBM and even Cray Research to use stock microprocessors in their new parallel machines. Therefore, it becomes more and more interesting to construct multithreaded parallel machines using stock processors that can tolerate latency.

As stock processors are not tuned for either parallel processing nor multithreading in general, they sometimes lack desirable features which should be added outside of the processor. Support for fast communication is a good example. It is essential in fine-grain multithreading to handle a large number of messages in an efficient way so that computation and communication could be overlapped. For this reason, some approaches are using a dedicated communication processor, separate from the main processor. However, no quantitative analysis has been reported on the effectiveness of employing the communication processor and its impact on the overall performance. In this paper, we explore three different node configurations in detail and compare their dynamic characteristics through the instruction-level simulation with six benchmark programs.

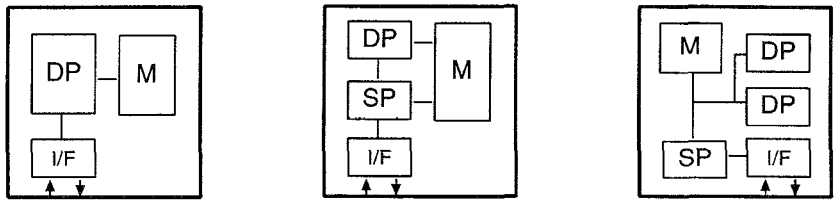
This paper is organized as follows. Section 2 introduces TAM, the multithreading model that this paper is based on. Section 3 shows our experimental methodologies and section 4 presents the results of the experiments. Section 5 concludes the paper.

2. Fine-grain Multithreading on Stock Processors

Our work is largely based on the TAM (Threaded Abstract Machine) model [4]. The TAM model allows us to construct multithreaded architectures using stock proces-

* This work was supported in part by the Korea Science and Engineering Foundation (KOSEF) under contract No. 951-0910-126-2.

0-8186-7901-8/97 $10.00 © 1997 IEEE



(a) S-CFG (b) D-CFG (c) C-CFG

Figure 1. Node configurations under consideration

sors because it does not assume any hardware support for multithreading. In the TAM model, all synchronization, scheduling and storage management are explicit in the machine language and under compiler control.

A TAM program consists of a collection of *code blocks*, which typically represents functions or loop bodies. A code block is compiled into a set of *inlets* and *threads*. Inlets are short message handlers that receive arguments and threads are sequences of code corresponding to the body of the code block. When a code block is invoked, a *frame* is allocated for storage of arguments, local variables, and a list of ready threads associated with the frame. When an inlet is executed, it typically writes the incoming value(s) into the frame and posts a dependent thread.

When a frame is activated, the list of ready threads is regarded as the *local continuation vector* (LCV). Any thread that forks other threads within the code block does so by placing them in the LCV. Threads are fetched and executed from the LCV until none remain, after which frame switching is performed. The set of threads executed in a single frame activation is called a *quantum*. The original TAM model provides special *enter* and *leave* threads that are intended to take advantage of the locality among multiple threads in a scheduling quantum, but the optimization has not been implemented in the TAM compiler.

Global data structures in TAM are based on I-structure [2] semantic, which provides synchronization on a per-element basis. Accesses to the data structures are split-phase and are performed via special instructions such as `ifetch` and `istore`.

There have been already many research projects to implement fine-grain multithreaded architectures using stock processors such as MTA(EARTH) [7], DAVRID [6], and *T-NG [1]. All those architectures were based on the execution model which is very similar to the TAM.

In this paper, we consider three node configurations as shown in figure 1. In S-CFG, each node consists of a data processor(DP), memory(M), and a network interface(I/F), while there is an additional synchronization processor(SP) in D-CFG. SP is a dedicated processor for the handling of incoming and outgoing messages and synchronizations. In S-CFG, both communication and synchronization tasks are handled by DP. Also if the load of SP is sufficiently low, it would be possible to associate two DPs with one SP, as shown in figure 1(c). Detailed simulation results will be presented in section 4.

3. Methodology

We have constructed an instruction-level simulator to evaluate the efficiency of three node configurations. The simulator was originally based on SPIM [8], an instruction-level simulator for MIPS instruction set, and was extended to parallel and multithreaded environments using a commercial event-driven simulator, SES/Workbench [10]. Our simulator accepts a program written in MIPS assembly language, and all the events generated during execution of the program are scheduled using SES/Workbench. The powerful graphical user-interface of SES/Workbench allows us to build and change architecture models in a relatively short time. The simulator can be used for parallel machines consisting of up to 256 nodes.

Table 1. Benchmark programs

Benchmarks	Arguments	TL0 lines	Assembly lines	Code size (KB)
`fib`	20	361	373	1.88
`qs`	500	2773	2984	16.65
`mmt44`	100.0	3964	4547	25.93
`paraffins`	17	10324	11580	66.90
`speech`	10240 30	6766	7523	45.98
`dtw`	100.0	3500	3878	24.62

Table 1 shows arguments and program sizes of six benchmark programs used in the experiments. `fib` computes the Fibonacci numbers using recursions. `qs` is a simple quick-sort program using accumulation lists. `mmt44` multiplies two matrices of floating-point numbers and sums

Table 2. Dynamic scheduling characteristics for 4 nodes in S-CFG and D-CFG.

Benchmarks	Conf.	Frames			Threads			Inlets	
		T_F	S_F	Q	T_T	Q_T	L_T	T_I	L_I
fib	S	21892	47100	2.15	291215	6.18	17.28	109456	21.95
	D	21892	73666	3.36	344350	4.67	14.89	109456	23.20
qs	S	3005	38739	12.89	224160	5.79	19.78	54447	19.10
	D	3005	45121	15.01	236927	5.25	18.18	54447	20.00
mmt44	S	58	45162	778.66	165529	3.67	199.14	508381	12.06
	D	58	72886	1256.66	220980	3.03	151.52	508381	12.44
paraffins	S	1810	178090	98.39	1541545	8.66	32.63	496358	14.87
	D	1810	343062	189.54	1871492	5.46	27.28	496358	17.49
speech	S	3619	323438	89.37	1153479	3.57	63.76	1602582	13.39
	D	3619	432557	119.52	1371720	3.17	51.48	1602582	13.85
dtw	S	801	779782	973.51	2740761	3.51	41.13	2068601	15.15
	D	801	1026785	1281.88	3234770	3.15	34.00	2068601	15.98

all elements of the product matrix. paraffins enumerates the distinct isomers of paraffins. speech determines cepstral coefficients for speech processing. And dtw implements a dynamic time warp algorithm used in discrete word speech recognition. These applications are originally written in Id, and compiled to TL0 programs by the TAM group[1]. TL0 is an intermediate language defined in the TAM model.

TL0 programs are first converted to MIPS assembly codes by a translator. The generated code consists of MIPS instructions, assembly directives and several system calls. System calls are used to transfer the control to the run-time system for some multithreading primitives. Previous approaches [5, 11] translated TL0 programs to C programs and ran them on commercial parallel machines such as CM-5. The proposed approach differs in that we have translated TL0 programs to native CPU instructions for the evaluation of fine-grain multithreading.

In our experiments, we assume users can access the network interface directly in memory-mapped style for low-overhead communication. The network has a uniform communication latency of 100 cycles. I-structures are distributed over nodes and LCV is implemented as a stack. Also polling mechanism is assumed instead of interrupt to accept messages when there is no SP.

4. Results

Figure 2 shows overall speedup for the benchmark programs from 1 to 32 nodes with respect to the total execution time of a single node in S-CFG. We see that C-CFG has very little benefit compared with D-CFG, while D-CFG almost doubles the performance of S-CFG in unsaturated region.

The speedup is closely related to the processor utilization, which is shown in figure 3. The average utilization of DPs decreases as the number of nodes increases, meaning that processor idle time increases because there are not enough threads to cover communication latencies. In figure 3, we can observe that the utilization of SP in C-CFG is roughly the same with that of SP in D-CFG except for fib and qs. This means that SPs in both configurations have almost the same loads playing as a bottleneck. Therefore, it is natural that the utilization of DP in C-CFG be almost half of that of DP in D-CFG, because two DPs are attached to one SP in C-CFG. Although the utilization of S-CFG is slightly higher than those of DP and SP in D-CFG, it does not necessarily imply S-CFG is more efficient than D-CFG because parts of processor cycles are spent to poll the network interface at the end of every thread in S-CFG.

Table 2 shows the dynamic characteristics of the benchmark programs for 4 nodes in S-CFG and D-CFG. T_F, T_T, and T_I in the table mean the total number of frames, threads and inlets, respectively. S_F stands for the total number of scheduled frames. The ratio of S_F and T_F means the average number of quanta in a frame, which is represented as Q. Q_T is the average number of threads in a quantum and L_T and L_I are the average length of a thread and that of an inlet in cycles, respectively.

The total number of frame scheduling is increased by about 48% in D-CFG by employing an SP. In S-CFG, a thread posted from an inlet can be put into LCV, if it belongs to the current frame. However it is impossible in D-CFG, because we assume LCV is not shared between DP and SP. Instead, thread continuations posted from inlets are recorded into the frame memory, which explains the slight

[1]They are freely available by anonymous ftp at ftp://ftp.cs.berkeley.edu/ucb/TAM/idtam-0.3.tar.Z

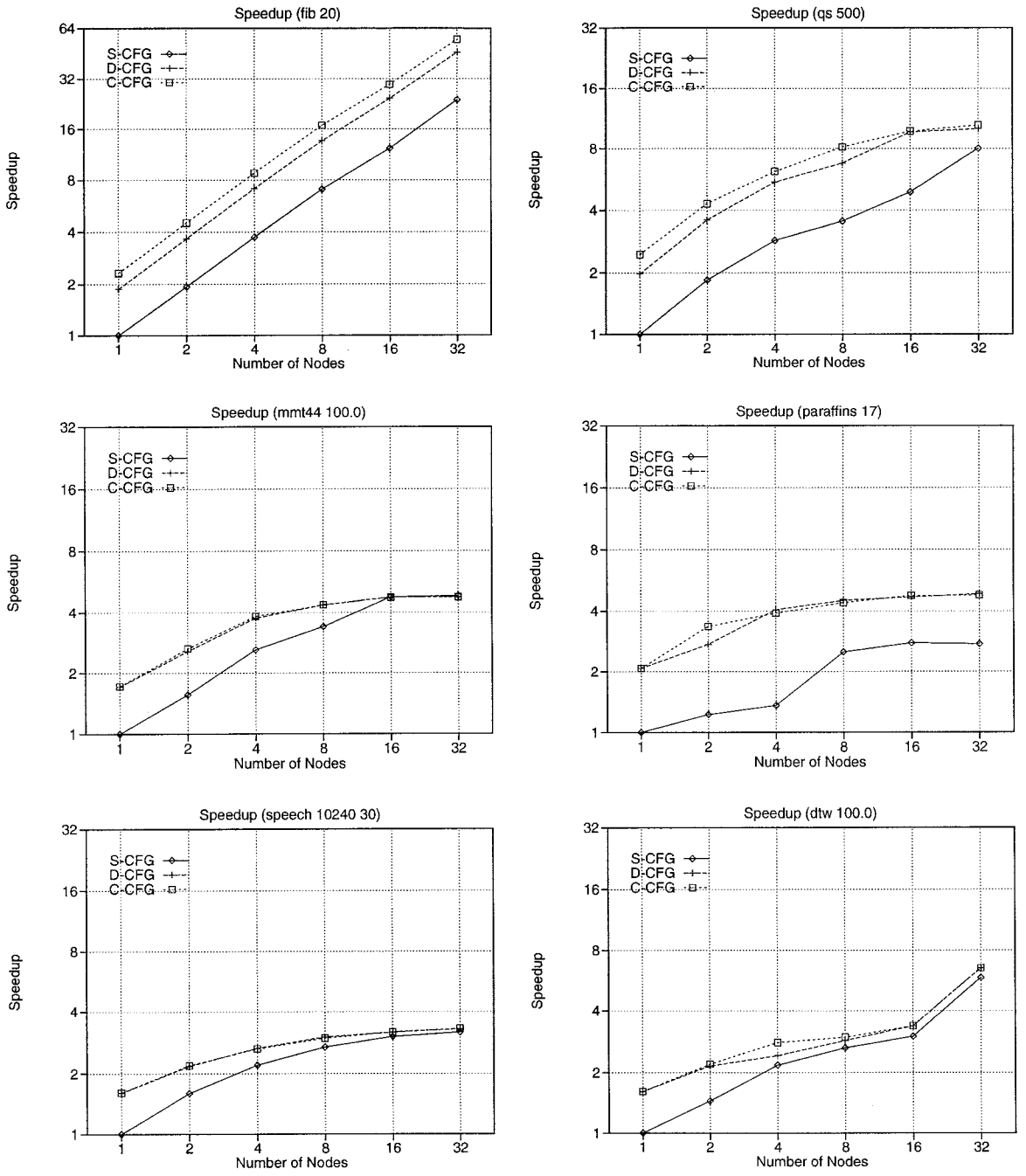


Figure 2. Speedup

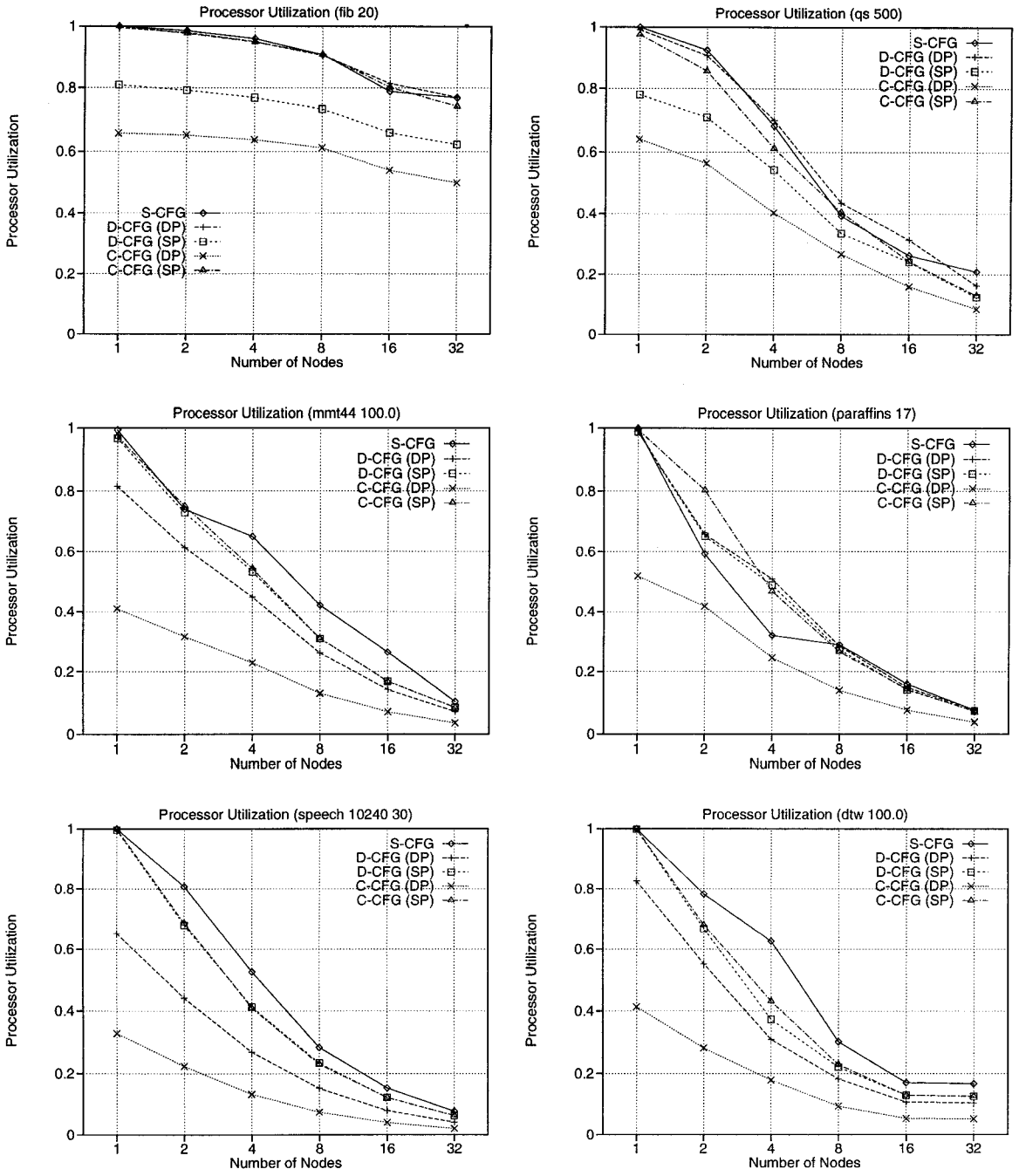


Figure 3. Processor utilization

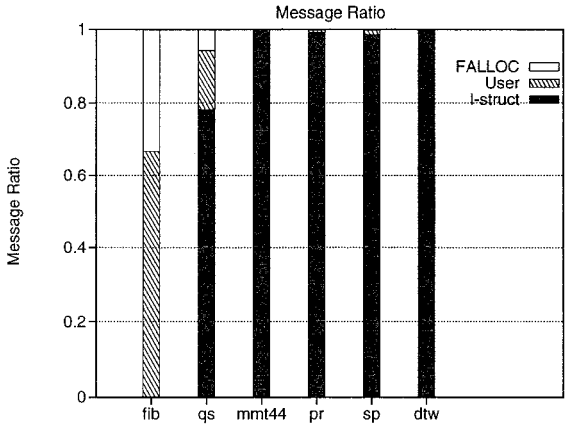


Figure 4. Classification of message traffics

increment of L_I in D-CFG. The increment in T_T is due to enter and leave threads, which are executed at the beginning and at the end of each frame activation.

We can take the ratio of the utilization of DP and SP in D-CFG as the ratio of computation and communication for the corresponding benchmark program. Higher utilization of DP (SP) means that the application is computation (communication)-bounded. From figure 3, `fib` and `qs` are examples of computation-bounded applications and the others of communication-bounded ones. This insight is apparent from table 2 where the number of messages ($= T_I$) of `fib` and `qs` is relatively small compared with other programs.

Figure 4 shows the classification of message traffics in each program. Messages used for allocating new frames, sending arguments or results, and accessing I-structures are labeled as `FALLOC`, `User`, and `I-struct`, respectively. In figure 4, it is clear that the large number of messages in `mmt44`, `paraffins`, `speech`, and `dtw` comes from I-structure accesses. Therefore we conjecture that I-structure accesses occupy a large fraction of total messages in most applications, and this communication intensiveness is a major factor that limits the overall speedup.

5. Concluding Remarks

In this paper, we have evaluated three different node configurations for fine-grain multithreading on stock processors. Our experiments have shown that attaching more than one DP to an SP as is done in C-CFG does not improve performance because SP becomes a bottleneck. However it seems reasonable to have a dedicated processor for handling communication and synchronization since such a configuration almost doubles the performance. It is important to balance computation and communication so that one should not be a bottleneck to each other.

In the experiments, the utilization of SP has been higher than or comparable to that of DP in D-CFG. Although our experiments assume the same power of DP and SP, there is an approach to use ASIC chips for the functionality of SP instead of stand-alone processors [7]. For such an approach to remain viable, the number of messages should be significantly reduced. The experimental results also suggest that we should pay special attention to the I-structure accesses, which tend to decide the total number of messages. It is also desirable to share LCV between DP and SP for D-CFG to reduce the frame scheduling overheads. But this would require complex hardwares due to the atomicity problem [9].

References

[1] B. S. Ang, Arvind, and D. Chiou. StarT the Next Generation: Integrating Global Caches and Dataflow Architectures. Technical Report CSG Memo 354, MIT, Feb. 1994.

[2] Arvind, R. S. Nikhil, and K. K. Pingali. I-Structures: Data Structures for Parallel Computing. Technical Report CSG Memo 269, MIT, Feb. 1987.

[3] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proc. 4th ACM SIGPLAN Symp. on Principles & Practices of Parallel Programming*, pages 1–12, 1993.

[4] D. E. Culler, S. C. Goldstein, K. E. Schauser, and T. von Eicken. TAM - A Compiler Controlled Threaded Abstract Machine. *J. of Parallel and Distributed Computing*, pages 347–370, Jun. 1993.

[5] S. C. Goldstein. The Implemenation of a Threaded Abstract Machine. Technical Report UCB/CSD 94-818, University of California at Berkeley, May 1994.

[6] S. Ha, J. Kim, E. Rho, Y. Nah, S. Cho, D. Hwang, and S. Han. A Massively Parallel Multithreaded Architecture: DAVRID. In *Proc. Int'l Conf. on Computer Design*, pages 70–74, 1994.

[7] H. H. J. Hum, K. B. Theobald, and G. R. Gao. Building Multithreaded Architectures with Off-the-Shelf Microprocessors. In *Proc. 8th Int'l Parallel Processing Symposium*, pages 288–297, 1994.

[8] J. R. Larus. SPIM S20: A MIPS R2000 Simulator. Technical Report #966, University of Wisconsin-Madison, 1990.

[9] D. Metz and B. Lee. Analyzing the Benefits of a Separate Processor to Handle Messages for Fine-grain Multithreading. In *Proc. 7th IEEE Symp. on Parallel and Distributed Processing*, 1995.

[10] Scientific and Engineering Software Inc. *SES/Workbench 3.0 User's Manuals*. 1995.

[11] E. Spertus and W. J. Dally. Evaluating the Locality Benefits of Active Messages. In *Proc. Fifth ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, 1995.