To appear in the 19th IEEE International Performance, Computing and Communication Conference-IPCCC2000, Feb. 20-22, 2000, Phoenix, Arizona, USA

Performance Analysis of the I-Structure Software Cache on Multi-Threading Systems *

Wen-Yen Lin and Jean-Luc Gaudiot
Dept. of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2563

José Nelson Amaral and Guang R. Gao
Dept. of Electrical and Computer Engineering
University of Delaware
Newark, Delaware

Abstract

Non-Blocking Multithreaded execution models have been proposed as an effective means to overlap computation and communication in distributed memory systems without any hardware support. Even with the capability of latency tolerance in these execution models, each remote memory request still incurs the cost of communication interface overhead. We therefore designed and implemented our I-Structure Software Cache system to further reduce communication overhead for non-blocking multithreaded execution.

In this paper, we present analytical models for the performances of a multithreading system with and without I-Structure Software Cache support. We compare our model's prediction with our experimental results on an existing multithreaded architecture platform. The analytical models allow us to predict at what ratio of communication latency/processing speed the implementation of I-Structure Software Cache becomes profitable for applications with different characteristics.

Keywords: Software Caches, Communication Interface Overhead, Multithreading Architecture, EARTH, ISSC.

1 Introduction

Multithreaded architectures have been proposed as a means to overlap computation and communication in distributed memory systems. By switching to the execution of other ready threads, the communication latency can be hidden from useful computations as long as there is enough parallelism in an application. *Split-phased transaction* [7, 16] schemes have been used in some multithreaded models, like TAM [4], P-RISC [13], and EARTH [5], to achieve communication latency tolerance. By splitting the remote memory access into two phases, *requesting* and *consuming*, the processor can continue executing other useful computations without waiting for the requested data to arrive after sending out the request. Along with other outstanding remote requests, the execution of the current thread or of other threads overlaps the communication latencies with useful computation.

In these models, a thread is activated when all the data elements it needs are available locally. Therefore, once a thread starts to execute, it executes to the end. In such a "Non-Blocking Multithreaded" model, once the execution of a thread terminates, no thread context needs to be saved before switching the execution to another active thread. The absence of context saving in the middle of a thread makes it possible to implement this kind of system with off-the-shelf processors [14]. Some examples of this kind of implementations are the TAM on CM-5 [2] and the EARTH on MANNA, Beowulf, and IBM SP2 [17].

However, a drawback of the non-blocking multithreaded execution model is that the locality of remote data is not fully exploited. If all remote requests are translated into split-phased transactions and the fetched data is not saved between usages, excessive inter-processor traffic will be generated. [1] The problem is compounded by the cost of sending and receiving network packets. Each packet may take from dozens to thousands of cycles to put on network depending on the design of the network interface and communicating protocols [3]. Even though multithreaded execution could tolerate the communication latency, processors still need to spend the time to issue the request and retrieve the data from network. In most platforms without dedicated processors to handle the network messages, this *communication interface overhead* cannot be overlapped with useful com-

*This research is supported in part by NSF grant # MIP-9707125 and INT-9815742

[1] To be fair, in most explicitly multi-threaded code, the programmer does save remotely fetched data in the local memory for future use. However, the code complexity to do so might be taxing on the programmer and the spatial data locality still can not be exploited.

putations. In some machines, a parallel environment is built on top of the TCP protocol and the communication interface overhead may be as high as hundreds of micro-seconds [9]. Even with some improved protocols, like Fast Sockets [15] and Active Messages [16], it still costs 40~60 micro-seconds to send a message to the network. For instance, in the multi-packet delivery implementation of Active Messages in the CM-5 machine, it costs 6221 instructions for sending a 1024-word message in the finite sequence [18]. Since all requests are actually sent to remote hosts through the network, all the sending and receiving requests incur the communication interface overhead and will result in high network traffic.

To overcome these problems, we designed and implemented an I-Structure Software Cache (ISSC) [10, 11, 12] to cache the split-phase transactions in a single-assignment memory system, like I-Structures [1]. By caching those split-phase transactions, the ISSC reduces the number of remote requests significantly, and hence, the amount of communication overhead incurred by remote requests is reduced. Indeed, with the capability of communication latency tolerance in multithreaded execution, the major benefit of ISSC comes from the saving from communication interface overhead. Our ISSC is a pure software approach to exploit the global data locality in non-blocking multithreaded execution without adding any hardware complexity. Therefore, in order for ISSC to deliver performance gains in non-blocking multithreaded systems, the overhead incurred by ISSC operations must be less than the amount of communication interface overhead saved by ISSC.

In this paper, we present an analytical model for the performance of a multithreading system with and without ISSC support. From this model, we could analyze the lower bound of communication interface overhead from which ISSC starts to yield performance gain in different benchmarks and platforms. The remainder of the paper is organized as follows. In the next section, we will introduce the design of our ISSC. In section 3.1, the analytical model and detailed performance analysis is described and we conclude this work in section 4.

2 The I-Structure Software Cache (ISSC)

In 1994 Dennis and Gao proposed the caching of elements of single-assignment data structures [6]. We designed and implemented an I-Structure Software Cache (ISSC) [10, 11, 12] to cache I-Structure elements on multi-threaded systems that support split-phased transactions. The ISSC takes advantage of the spatial and temporal localities of memory operations in I-Structure memory systems. The ISSC system works as an interface between user applications and the network interface. It intercepts all the read operations to the I-Structure. A remote memory request is sent out to the remote host only if the requested data is not available on the software cache of the local host. We explore the spatial locality in the references to the I-structure through a blocking mechanism. Instead of requesting a single element of the structure, an entire block of data including the requested element is requested to the node that hosts the I-structure.

2.1 Features of ISSC

The detailed design of ISSC and its implementation are described in [12, 20]. The most relevant features of our I-Structure software cache can be summarized as follows:

- **Write-direct policy.** A simple *write direct* policy is adopted in our design and therefore, there is no caching for write operations. It ensures the legality of write operations for a single assignment memory.

- **Cache advance.** In our design, a cache line is allocated in the software cache before the new request for the block of data containing the requested element is sent to the remote host. The original request and subsequent requests for data elements in the same cache block can be deferred in the pre-allocated cache line before the requested data block is brought back from remote host.

- **Deferred read sharing.** To ensure the requests which have been deferred in local software cache to receive their data, the new request for block of data is shared by all the empty data elements of the requested data block in the I-Structure by appending the request to all empty locations. Therefore, the data could be forwarded to the requesters as soon as they are generated.

- **Centralized deferred requests and distributed deferred reads.** A simple "centralized" method is used for the implementation of the queues of deferred requests in the I-Structure. In fact, the length of the queue of deferred requests for each element in the I-Structures is bounded by the number of nodes in the system. This is because only one request is sent from each node to the host node. Future deferred reads are kept locally in the node.

Operation	Local	Remote
Get_Sync	141	348
Fun. Call	250	451
I_READ_F	317	492
ISSC hit	479	—
ISSC miss	2693	—
ISSC deferred	1354	—

Table 1: Latency of EARTH and ISSC operations on EARTH-MANNA-SPN, measured in number of cycles (1 cycle = 20 ns).

2.2 ISSC implementation on EARTH-MANNA

We implemented *ISSC* [20] in the Threaded-C [8] language for EARTH systems. The EARTH [17, 5], Efficient Architecture for Running Threads, is an architecture and program execution environment that defines a fine-grain non-blocking multi-threading model.

Our studies are based on an implementation of EARTH on the MANNA machine. MANNA [19] is a 20 node, 40 processor machine. Each node has two Intel i860 XP processor running at 50 MHz with 32 MB memory and is interconnected with other nodes through a crossbar switch network. The MANNA machine is a research platform of which only a few were constructed. With the full control of network interface in MANNA machine, the implementations of inter-node communication and synchronizations are very efficient as demonstrated by the measurements presented in this section. We measure the latency of some EARTH and ISSC operations for the EARTH-MANNA-SPN machine. EARTH-MANNA-SPN is an implementation of the EARTH model on the MANNA machine in which only one processor is used in each node [17].

Table 1 lists the latency of some EARTH and ISSC operations in the MANNA platform used in the analytical model. In a local measurement all operations are within a processor, while in a remote measurement, all operations are issued to other nodes through network. The EARTH operations measured in Table 1 include a `get_sync` operation in which thread 1 requests a word of data from thread 2 and thread 2 synchronizes thread 1 when the data arrives; and function calls which represent the invocation of a threaded function either in the same node or in a remote node.

At the bottom of Table 1 are the measured latency of ISSC operations and of the basic I-Structure read function, `I_READ_F`. The measurement starts from thread 1 invoking the I_READ_F function in I-Structure node either in the same node or in a remote node until the I_READ_F function finished and synchronizing thread 1 when the data arrives. `ISSC hit` measures the invoking of an I_READ_F for a remote data, finding the requesting data in local software cache and synchronizing the requesting thread with the data found in software cache. `ISSC miss` is the case that the entire surrounding data block is not found in the software cache and a new request for the whole block is issued to a remote node, and finally the requested data along with the whole data block are sent back from remote node and the synchronization is done. Notes that, this measurement is made by issuing multiple requests in a pipeline fashion. Therefore the time spent on the remote node is overlapped with other issues of requests and only the time spent in local node is measured. `ISSC deferred` is the case that the surrounding data block already allocated in the local software cache however the requested data element is not there yet. The original request is therefore deferred in the software cache until the requested data is available along with entire data block or sent back individually from remote I-Structure node. The same measurement as *ISSC miss* is done to ensure that no idle time and remote operation time is measured.

The difference between local and remote cases of I_READ_F denotes four times of the communication interface overhead. It includes one for the requester sending the request, one for the I-Structure node receiving the request, one for I-Structure node sending the data back and finally one for the requester receiving the data. The one-way communication interface overhead takes only 175/4 processor cycles (0.825 μs). This measurement indicates that the inter-node communication in MANNA machines is very efficient when compared with network of workstations.

3 The Analytical Models

3.1 Performance modeling

The execution time of benchmarks in a non-blocking multithreaded multiprocessor system written in plain Threaded-C [8], $T_{threaded-c}$, system with I-Structure support, T_{IS}, and system with both I-Structure and I-Structure Software Cache support, T_{ISSC} could be modeled in the following equations:

$$
\begin{aligned}
T_{threaded-c} &= T_B + (N_L + N_R)O_r + N_R 2(C_o + C_{oa}) \\
T_{IS} &= T_B + N_L O_I + N_R O_r + N_R 2(C_o + C_{oa}) \\
T_{ISSC} &= T_B + N_L O_I + N_R R_{hit}(1 - R_{d-hit})O_{hit} \\
&+ N_R R_{hit} R_{d-hit} O_{def} + N_R(1 - R_{hit})O_{miss} \\
&+ N_R(1 - R_{hit})2(C_o + C_{oa})
\end{aligned}
$$

Where T_B is the base for our analytical model which is the execution time of the benchmark on a fine grain multi-threaded machine without I-Structures and the cost of split-phase memory accesses is deducted. We could classify the parameters used in our models into two category, *benchmark-related parameters* and *platform-related parameters*:

Benchmark-related parameters:

N_L:	Number of local reads
N_R:	Number of remote reads
R_{hit}:	Cache hit ratio on remote reads
R_{d-hit}:	Cache deferred hit ratio

Platform-related parameters:

C_o:	One-way communication interface overhead (original)
C_{oa}:	One-way communication interface overhead (add-on)
O_I:	Local I-Structure read service time
O_r:	Read request invoking time
O_{hit}:	ISSC hit service time
O_{miss}:	ISSC miss service time
O_{def}:	ISSC deferred hit service time

Where R_{d-hit} is the ratio between the cache hits that have been deferred and the total number of cache hits. The higher R_{d-hit} is, the poor temporal data locality is in the application. The C_o and C_{oa} are defined as one-way communication interface overheads which are only incurred in either sending or receiving network data, but not both. O_r is the overhead of invoking a split-phased read request. The others are the overheads of I-Structure and ISSC operations. Notice that the request invoking time is already included in O_I, O_{hit}, O_{miss}, and O_{def}. Also, in these models O_{miss} does not include communication interface overhead.

In the development of the analytical model, we assume owner computation rule. Therefore, all the write operations are performed locally and incur no communication overhead. We also assume that the I-structure arrays are evenly distributed across the nodes. Therefore that the jobs are also evenly distributed. We assume the same basic execution time, T_B for all three versions of the system. In fact, T_B in T_{ISSC} should be smaller than the ones in $T_{threaded-c}$ and T_{IS} because caching remote memory requests decreases the average turn-around time for all the requests and as a result, it increases parallelism and processor utilization. However, this assumption in T_{ISSC} provides the upper-bound of the execution time for the system with ISSC. In our implementation, only remote reads are cached in ISSC. Hence, those local I-Structure reads in T_{ISSC} still need the I-Structure read service in local node. In these models, the remote costs for $T_{threaded-c}$ and T_{IS} are $N_R 2(C_o + C_{oa})$ and for T_{ISSC} is $N_R(1 - R_{hit})2(C_o + C_{oa})$ which only include the communication overhead incurred in the local node. The overheads in remote node are actually hidden by the multithreaded execution.

3.2 Model verification

Indeed, with the capability of communication latency tolerance in multithreaded execution, the major benefit of ISSC comes from the saving from communication interface overhead. To find out when the saving of communication interface overhead compensates the ISSC operation overhead and hence ISSC starts to yield performance improvement, this critical point could be derived from $T_{ISSC} < T_{threaded-c}$,

$$(N_L + N_R)O_r + N_R 2(C_o + C_{oa}) > N_L O_I$$
$$+N_R R_{hit}(1 - R_{d-hit})O_{hit} + N_R(1 - Rhit)2(C_o + C_{oa})$$
$$+N_R R_{hit} R_{d-hit} O_{def} + N_R(1 - R_{hit})O_{miss}$$

$$\Rightarrow \frac{N_R}{N_L+N_R} R_{hit}(2C_o + 2C_{oa}) > \frac{N_L}{N_L+N_R} O_I$$
$$+\frac{N_R}{N_L+N_R} R_{hit}((1 - R_{d-hit})O_{hit} + R_{d-hit} O_{def})$$
$$+\frac{N_R}{N_L+N_R}(1 - R_{hit})O_{miss} - O_r \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

The meaning of Equation 1 is quite straight forward. The condition for ISSC starts to improve the system is that the communication interface overhead saved by ISSC (left hand side of the equation) should be greater than the I-Structure read service time required for local access plus ISSC operation overhead minus the read request handling time in the original system (right hand side of the equation).

In our ISSC implementation [20] on EARTH-MANNA systems, we measured the system performance when both I-Structures and ISSC are used in Threaded-C language on a set of selected benchmarks: dense matrix multiplication, Conjugate Gradient, Hopfield network, and sparse matrix multiplication. To compare the performance of the software cache with the original system, we implemented three versions of codes for each benchmark: A *plain Threaded-C* code, a Threaded-C code using the I-Structure library, *Threaded-C+IS* and a Threaded-C code using both the I-structure library and the I-Structure Software Cache (*ISSC*), *Threaded-C+ISSC*.

We ran our experiments on a 16 node EARTH-MANNA system with adding a variable synthetic communication interface overhead on top of existing overhead. In Table 2, we list the benchmark-related parameters which are collected from our experiments and

Parameters	Benchmarks			
	Dense M.M.	C.G.	Hopfield	Sparse M.M
N_L	139328	614	512	19140
N_R	123840	9210	7680	234784
R_{hit} (%)	98.35	93.65	99.22	99.55
R_{d-hit} (%)	20.00	51.80	100.00	21.20

Table 2: Benchmark-related Parameters

Parameters	C_o	O_I	O_r	O_{hit}	O_{miss}	O_{def}
micro-second	0.875	6.34	2.82	9.58	51.54	27.08

Table 3: Platform-related Parameters Measured from MANNA machine

the platform-related parameters of MANNA machine, measured in Section 2.2, are listed in terms of μs in Table 3. The critical points, where ISSC starts to deliver performance gains for the selected benchmarks, measured from the experiments are listed in the top row of Table 4.

In order to verify the analytical models, we plug in the benchmark-related parameters listed in Table 2 and the platform-related parameters listed in Table 3 to Equation(1) to derive the critical point for each benchmark from our models. The derived critical points are listed in the bottom row of Table 4. Our analytical model for T_{ISSC} defines the upper bound of the execution time. Therefore, the cross-point derived from Equation 1 is the lower-bound of communication interface overhead from which ISSC starts to improve system performance. For example, if the point derived from our models is 10μs, for this upper-bound estimation of T_{ISSC}, we could say that as long as the communication interface overhead is larger than 10μs, our ISSC is going to improve the performance. Values of these cross-points derived from our analytical models are greater than but close to the values we measured in our experiments shown in Table 4 except in Hopfield. This is because the synchronization of the activation updates after each time stamp yields partial sequential behavior. In this case, the basic execution time in T_{ISSC} is much smaller than in $T_{threaded-c}$. Therefore the cross-point we predicted is much larger than what we measured.

Benchmarks	Dense M.M.	C.G.	Hopfield	Sparse M.M
Measured	6.3	9.2	6	3.2
Derived	6.7	9.4	11.5	4.6

Table 4: Critical Points (in *mus*) of selected benchmarks

3.3 Performance Predictions

In this section, we introduced our analytical models for the multithreading system with and without I-Structure software cache support and we verified these models with our experiment results based on EARTH-MANNA machine. With these models, we could predict the lower bound of communication interface overhead from which ISSC starts to yield performance gain in different kind of benchmarks and platforms.

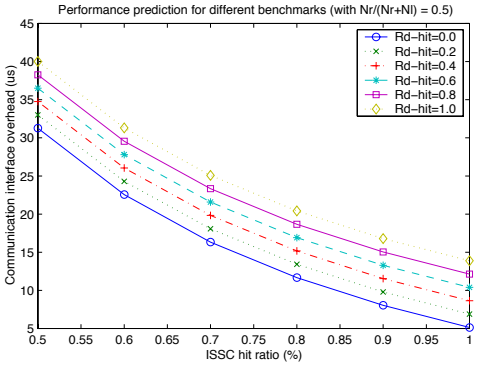

Figure 1: Performance prediction for different benchmarks

By using these models, for a fixed platform parameters (like plug in the parameters measured from EARTH-MANNA) and varied benchmark-related parameters, we could estimate the value of communication overhead where Threaded-C+ISSC starts to outperform pure Threaded-C for the benchmarks with different characteristics. Figure 1 shows these cross-points of different kinds of benchmarks by varying the cache hit ratio and deferred hit ratio while assuming only half of memory requests are issued to remote nodes. From this figure, we could see that even in those benchmarks with poor locality ($R_{hit} = 0.5$ and $R_{d-hit} = 1.0$), ISSC still yield performance gain for communication interface overhead greater than 40μs, which is still faster than most of the network implementation in network of workstations. For those benchmarks with extremely good locality, *i.e.* more than 98% of cache hit ratio with 0 deferred hit, ISSC starts to improve the system for the communication overhead as low as 5μs.

Some researcher dedicate their work on communication optimizations to reduce the number of remote memory accesses. This kind of optimizations are based on the static analysis of the program behavior which is different from exploiting the data locality during the run-time by the caches. However, ISSC could still yield performance gain in the benchmarks compiled

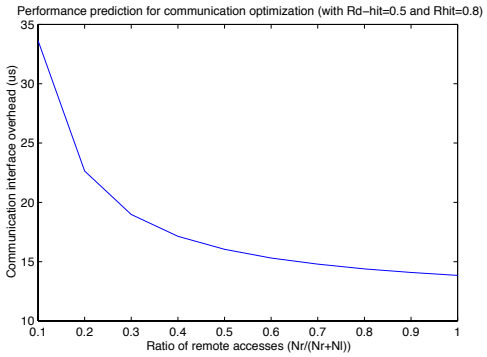

Figure 2: Performance prediction for communication optimization

with the communication optimization techniques. In Figure 2, we vary the ratio of remote memory requests to the total number of memory requests. We find out that even in an application with only 10% of memory accesses are remote and moderate cache hit ratio ($R_{hit} = 0.8$ and $R_{d-hit} = 0.5$) ISSC still improves the system at 33.5μs of communication overhead.

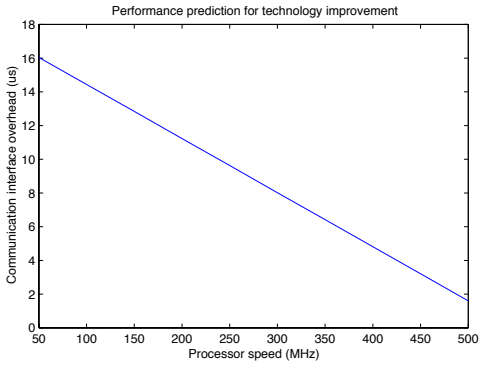

Figure 3: Performance prediction for technology improvement

As the speed of processors becomes faster and faster, the gap between the computation and communication latencies become larger and larger. Because, our ISSC is a pure software implementation, the ISSC operation overhead decreases proportional to the increase of processor speed. In Figure 3 we vary the platform-related parameters based on 50MHz MANNA processor by increasing the speed of processors for an application with 50 % of remote memory accesses, 80% cache hit ratio, and 50% deferred hit ratio. From this curve, we could predict that if we have a 500MHz processor available, which is already there on the market, the cross-point drops to less than 2μs. In this case, ISSC could almost yield performance gain on any parallel machine.

4 Conclusions

Do software caches really work? In this paper, we demonstrated a software implementation of I-Structure cache, *i.e. ISSC*, can deliver performance gains for most distributed memory systems which don't have extremely fast inter-node communications, such as network of workstations [3, 9, 15, 18].

ISSC caches values obtained through split-phase transactions in the operation of an I-Structure. It also exploit spatial data locality by clustering individual element requests into block. Our experiment results show that the inclusion of ISSC in a parallel system that provides split-phase transactions reduces the number of remote memory requests dramatically and reduces the traffic in the network. The most significant effect to the system performance is the elimination of the large amount of communication interface overhead which is incurred by remote requests.

We developed analytical models for the performance of a distributed memory multithreading system with and without I-Structure Software Cache support. We verified these models with our experiment results on an existing multithreaded architecture, EARTH-MANNA. These models consist of two sets of factors, platform-related and benchmark-related. Platform-related parameters are those latencies incurred by remote memory requests and ISSC operations. Benchmark-related parameters are the characteristics of applications, such as number of remote and local memory accesses and data locality. By finding the cross-point of two execution time curves, which have the communication interface overhead as variable, of the systems without and with ISSC, we could find when ISSC starts to yield performance improvement for different benchmarks and platforms. Through systematic analysis, we show that ISSC delivers performance gains for a wide range of applications in most of the parallel environments, especially in network of workstations.

References

[1] Arvind, R. S. Nikhil, and K. K. Pingali. I-Structures: Data Structures for Parallel Computing. *ACM Transactions on Programming Languages and Systems*, October 1989.

[2] David E. Culler, Seth Copen Goldstein, Klaus Erik Schauser, and Thorsten von Eicken. Empirical study of a dataflow language on the CM-5. In G.R. Gao, L. Bic, and J-L. Gaudiot, editors, *Advanced Topics in Dataflow Computing*

and Multithreading, pages 187–210. IEEE Press, 1994.

[3] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993.

[4] D. E. Culler, A. Sah, K. E. Schauser, T. von Eicken, and J. Wawrzynek. A compiler-controlled threaded abstract machine. In *Proceedings of ASPLOS-IV*, April 1991.

[5] H. H.J. Hum, O. Maquelin, K. B. Theobald, X. Tian, X. Tang, G. Gao, P. Cupryk, N. Elmasri, L. J. Hendren, A. Jimenez, S. Krishnan, A. Marquez, S. Merali, S. S. Nemawarkar, P. Panangaden, X. Xue, and Y. Zhu. A Design Study of the EARTH Multiprocessor. In *PACT 95*, June 1995.

[6] J. B. Dennis and G. R. Gao. On Memory Models and Cache Management for Shared-Memory Multiprocessors. CSG MEMO 363, Laboratory for Computer Science, MIT., March 1995.

[7] J. Hicks, D. Chiou, B. S. Ang, and Arvind. Perfornamce Studies of Id on Monsoon Dataflow System. *Journal of Parallel and Distributed Computing*, pages 273–300, 1993.

[8] J.N. Amaral, Z. Ruiz, S. Ryan, A. Marques, C. Morrone, and G.R. Gao. Portable Threaded-C Release 1.1. Technical note 05, Computer Architecture and Parallel System Laboratory, University of Delaware, September 10 1998.

[9] K. Keeton, T. Anderson, and D. Patterson. LogP Quantified: The Case for Low-Overhead Local Area Networks. In *Hot Interconnects III: A Symposium on High Performance Interconnects*, August 1995.

[10] W.Y. Lin and J-L. Gaudiot. I-structure Software caches - A split-phase transaction runtime cache system. In *Proceedings of the 1996 Parallel Architectures and Compilation Techniques Conference*, Oct. 1996.

[11] W.Y. Lin and J-L. Gaudiot. Exploiting Global Data Locality in Non-Blocking Multithreaded Architectures. In *Proceedings of the Third International symposium on Parallel Architectures, Algorithms and Networks*, Dec. 1997.

[12] W.Y. Lin and J-L. Gaudiot. The Design of An I-Structure Software Cache System. In *Workshop on Multithreaded Execution, Architecture and Compilation, 1998. Held in conjunction with HPCA-4*, Feb. 1998.

[13] R. S. Nikhil and Arvind. Can dataflow subsume von Neumann computer? In *Proceedings of ISCA-16*, May-Jun 1989.

[14] O. C. Maquelin, H. H.J. Hum, and G. R. Gao. Costs and Benefits of Multithreading with Off-the-Shelf RISC Processors. In *Proceedings of EURO-PAR'95*, August 1995.

[15] S. Rodrigues, T. Anderson, and D. Culler. High-Performance Local Area Communication With Fast Sockets. In *USENIX 1997 Annual Technical Conference*, Jan 1997.

[16] T. von Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauser. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 256–266, May 19–21, 1992.

[17] Kevin B. Theobald. EARTH - an Efficient Architecture for Running THreads. Ph.d thesis, School of Computer Science, McGill University, Montreal, Québec, 1999.

[18] V. Karamcheti and A. Chien. Software overhead in messaging layers: Where does the time go? In *Proceedings of the 6th ACM International Conference on Architectural Support for Programming Languages and Systems (ASPLOS VI)*, Oct. 5-7, 1994.

[19] W.K. Giloi, U. Bruning and W. Schroder-Preikschat. MANNA: Prototype of a Distributed memory Architecture With Maximized Sustained Performance. In *Proc. Euromicro PDP96 Workshop*, 1996.

[20] W.Y. Lin, J. Nelson, J-L Gaudiot, and G. Gao. Caching Single-Assignment Structures to Build a Robust Fine-Grain Multi-Threading System. In *International Parallel and Distributed Processing Symposium (IPDPS2000)*, May 2000.