Caching Single-Assignment Structures to Build a Robust Fine-Grain Multi-Threading System

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, DE 19716

Abstract

We present the design, implementation, and evaluation of single assignment data structures and of a software controlled cache in an existing multi-threaded architecture platform – the Efficient Architecture for Running Threads (EARTH). The software-controlled cache (ISSC) exploits temporal and spatial locality of EARTH split-phased memory transactions for single-assignment memory references. Our experimental evaluation indicates that the caching mechanism for single-assignment storage makes the EARTH memory system more robust to variations in the latency of memory operations. As a consequence the system can be ported to a wider range of machine platforms and deliver speedup for both regular and irregular application.

Keywords: *Single Assignment, Split-Phase Transactions, Software Cache, Fine-Grain Multi-Threading, EARTH.*

1. Introduction

The design of a fine grain multi-threading system can be evaluated with many different metrics. What is the cost of switching between threads? How many instructions must a thread run, on average, in order to enable the implementation of an efficient latency hiding mechanism, and keep the processing units usefully busy? How can the system automatically balance the computation load across many processing nodes? How much latency can the system tolerate without significant performance penalty?

The Efficient Architecture for Running Threads (EARTH) [8, 15] is an architecture and program execution environment that defines a fine-grain multi-threading model. Multi-threading programs for EARTH are written in Threaded-C, an explicitly multi-threaded extension of the C language. In this paper, we demonstrate that the addition of both a single-assignment structure (I-Structure) and a temporary storage for these structures (I-Structure Software Cache) to the EARTH fine-grain multi-threading system improves the robustness of the system both in relation to latency tolerance and application demands on the memory system. This robustness is reflected in better speedup curves for machines with higher latency for remote operations.

The remainder of the paper is organized as follows. Sections 2 discuss modern multi-threading system and their split-phase transactions. Section 3 presents our implementation of I-structures and I-Structure Software Caches (ISSC). Section 4 is a detailed performance study based on a set of real benchmarks. Section 5 presents related work and we conclude this work in Section 6.

2 Fine Grain Multi-Threading

Modern multi-threaded systems can be classified into two broad classes according to the granularity of the threads that they can efficiently support while yielding good performance: coarse grain multi-threading and fine grain multi-threading. Typically in a coarse grain multi-threading system, the thread switching mechanism involves interactions with the operating system, and there is a limited number of light-weighted processes to which threads must be bound. In a coarse grain multi-threading system, a thread can be viewed as a refinement of an operating system process. In contrast, in a fine grain multi-threading system: (1) the unit of computation is a collection of instructions grouped in a code block; (2) the system does not impose limits on the number of threads that can be active at a given time; (3) the system does not require binding to any sort of limited resources; and (4) the thread

switching mechanism is quite efficient and does not involve the operating system, the switching mechanism typically requires that only a small amount of state information be saved in each switching. In a fine grain multi-threading system, a thread can be viewed as the coarsening of an instruction.

EARTH, the fine grain multi-threading system studied in this paper, is derived from the data-flow model of computation. In the classical strict data-flow model an instruction is enabled for execution when all its operands are available [6, 7]. To enforce this enabling condition, the instructions that produce operands must be able to send a synchronization signal to all the instructions that will consume their results. This model proved unwieldy for the implementation of machines based on current standard off-the-shelf hardware and compiler technology. In EARTH, the unit of computation is not an instruction, but a code-block formed by many instructions. An instantiation of the code-block running on a processing node is called a *fiber*, and multiple code-blocks are grouped into *threaded functions*. A successful program written in Threaded-C, the programming language for EARTH, will produce enough fibers to keep the local processor busy while remote computations and data fetching operations are performed.

A cornerstone of the EARTH model is the mechanism that enables the superposition of local computation and remote operations: the *split-phase transaction*. A remote operation usually involves a long and/or unpredictable latency. Whenever a remote operation is requested, the statement requesting that the operation be performed is issued in one fiber, the requesting fiber, and the statement that depends on the result of the remote operation is issued on a different fiber, the consuming fiber. The issue of remote operation contains the *storage address* for the result and the *synchronization slot* for the consuming fiber. After issuing the remote operation, the requesting fiber may continue with other operations or it may terminate if there are no more operations to be executed by the the requesting fiber.

In the mean time, when the operation is completed in the remote node, the results are sent back to a pre-specified *storage address* in the local node and a signal to a pre-specified *synchronization slot* informs the consuming fiber about the arrival of the remote operation result. When all the data needed by the consuming fiber is available locally and all the synchronization events are matched, the consuming fiber becomes ready for execution. Using this kind of split-phased transactions, the long latencies of remote operations can be overlapped with useful local computation.

3 Implementing I-Structure and its caching scheme on EARTH systems

Our motivation to introduce a single assignment structure, such as I-Structures [1], in Threaded-C stems from the observation that the use of such structures significantly reduces the number of synchronization operations required by some programs. The single-assignment characteristic of I-structures eliminates the need for consistency related network operations when these structures are enhanced with temporary storage buffers. The former makes it easier to code problem solutions in Threaded-C, and the latter makes it easier to implement software caches for I-structures.

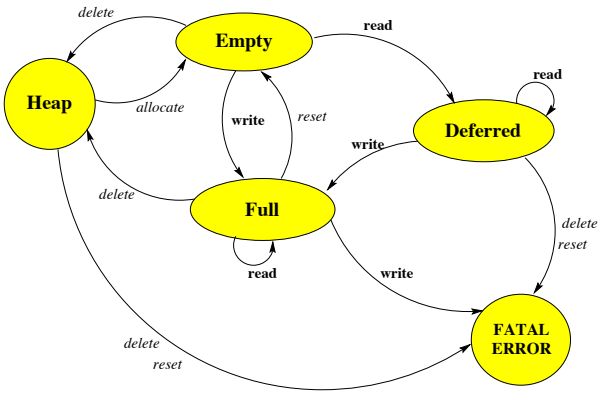

Figure 1. State Transition Diagram for the I-Structure Implementation

An I-structure is defined as an array of elements where the elements could be any type of data, such as integers, floating points, characters, or even data structures as in our implementation. Each element of the array can be in one of three states: *empty*, *full*, and *deferred*, and it can only be written once, but read many times. Figure 1 shows the state diagram of an I-structure. When the I-structure is created, it is allocated from heap memory and all the elements of the array are empty. If a read occurs before the write, the element goes into the deferred state and the read operation is kept in a queue associated with that element. Subsequent reads are also queued. When a write to an empty element occurs, the value is written and the element becomes full. If the element was in the deferred state, all the reads that were queued for that element are serviced before the writing operation is completed, and the element goes into the full state. A read to a full element returns immediately with the value previously written. A write to a full element is considered a *fatal error* and causes the program to terminate.

3.1 I-Structure Software Cache implementation

Split-phased transactions for remote data memory accesses provide the ability to tolerate communication latency in a multi-threaded system. The data obtained through a split phase transaction is managed by the programmer, and is not automatically cached by the system. Therefore if repeated requests for the same data are issued, they will be sent through the network to the source of the data requested.

In 1994 Dennis and Gao proposed caching the elements of single-assignment data structures [5]. We designed and implemented an I-Structure Software Cache (ISSC) [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 single assignment property of the I-Structure memory system enables the implementation of the ISSC as a *software cache* without any hardware support. The ISSC 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 elements of 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.

We implement *ISSC* in the Threaded-C language for EARTH [8, 15] systems. The layout of our software cache is the one of a set-associative cache. Set-associative software caches have faster cache entry searching time than fully associative caches and better cache utilization than direct mapped caches. The caching address consists of the node number of the host node, the I-Structure I.D. and the index of the element for which a read is requested. Upon receiving a read request, the caching address is mapped to a set by a hash function, and a software search is performed to see if there is a match for the address in the set. In our simulation studies [11], we determined that a cache block of 8 data elements would yield reasonable cache hit ratio. Therefore, in our experiments, discussed in section 4, we use a cache block size of 8 and implemented the software cache with 256 sets and 8 cache blocks within each set. That would be $16K$ elements in the cache.

4 Performance Measurements

Before we study the effectiveness of our implementations of both I-structures and ISSC on selected benchmarks, we measured the latency of some basic EARTH, I-Structure, and ISSC operations. Our experimental results were obtained on a 20-node MANNA machine. The MANNA [16] machine is a research platform of which only a few were constructed. 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. In order to obtain a fair comparison with other existing parallel machines that only have a single processor in each node, we performed our experiment on the EARTH-MANNA-SPN configuration. EARTH-MANNA-SPN is an implementation of the EARTH model on the MANNA machine in which all the functions are performed by a single processor [15].

Operation	Local	Remote
Get_Sync	141	348
Fun. Call (1)	250	451
Fun. Call (18)	410	628
I_READ	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).

Table 1 displays the latency of basic operations in number of cycles measured on the MANNA platform. The EARTH operations measured include a `Get_sync` operation in which a word of data (local or remote) is requested through split-phased transaction and function calls with 1 and with 18 parameters, that represent the invocation of a threaded function either in the same node or in a remote node. I_READ is a basic I-Structure read function for an element. At the bottom of Table 1 are the measured latency of ISSC operations. Note that ISSC deferred is the case in which a read hits the cache block that has been requested and for which no data is available at the moment. Notice that ISSC operations will only be performed when remote data are requested, therefore, no latency for local operations were measured.

The difference between local and remote cases of the I_READ operation represents the communication interface overhead. With the full control of network interface in MANNA machine, the implementations of inter-node communication and synchronizations are very efficient. The remote operation takes only 175 processor cycles (3.5 μs) more than the local one to

Number of Nodes	Benchmarks			
	Dense M.M.	C.G.	Hopfield	Sparse M.M
2	99.71	93.70	99.90	99.92
4	99.52	93.69	99.80	99.87
8	99.13	93.52	99.61	99.76
16	98.35	92.92	99.22	99.53

Table 2. ISSC cache hit ratios(%)

Number of Nodes	Benchmarks			
	Dense M.M.	C.G.	Hopfield	Sparse M.M
2	99.71	93.70	99.90	99.92
4	99.52	93.69	99.81	99.87
8	99.13	93.53	99.61	99.76
16	98.35	92.98	99.22	99.53

Table 3. Percentage of reduced remote memory requests by ISSC(%)

be completed, and the one-way network interface overhead takes only 175/4 processor cycles (0.825 μs). This measure indicates that the inter-node communication in MANNA machines is very efficient when compared with networks of workstations which may be as high as hundreds of micro-seconds [9].

4.1 Cache Performance

To measure the performance of I-Structures and ISSC, we selected four different benchmarks: *Dense matrix multiplication*, *Conjugate Gradient*, *Hopfield network*, and *Sparse matrix multiplication*. *Dense matrix multiplication* is a simple minded, nonblocking algorithm that multiply two 128x128 dense matrices. *Conjugate Gradient* from the NAS benchmark suite [2] solves 256 linear equations with 256 unknown variables. *Hopfield Network* is a kernel benchmark based on the Hopfield Network often used in combinatorial optimization problems. *Sparse matrix multiplication* is an application with irregular data access pattern. The size of sparse matrices is 256x256.

Table 2 shows the cache hit ratios of the four benchmarks in our experiments. For three of the four benchmarks (except conjugate gradient), more than 99% of cache hit ratios could be achieved, and even in the conjugate gradient algorithm, that has poor temporal data locality, 93% of cache hit ratio could be achieved. Table 3 shows the percentage of remote memory requests been reduced by ISSC. In all the cases, at least 93% of the original remote memory requests are eliminated out by the I-Structure Software Cache.

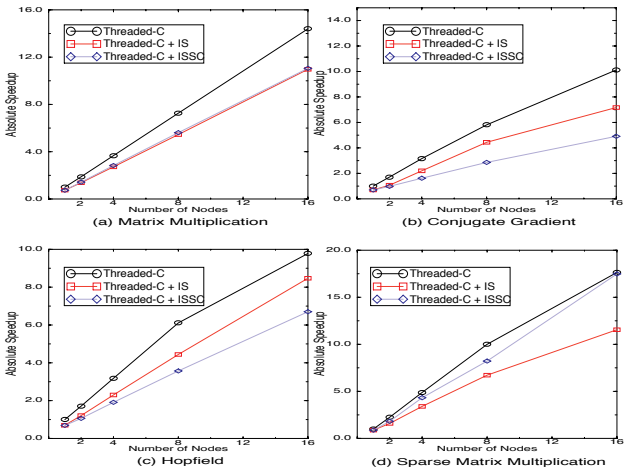

Figure 2. Speedup in the MANNA machine.

4.2 Robustness to Latency Variation

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*.

The speedups reported in Figures 2, 3, and 4 are computed in relation to the *plain Threaded-C* version on a single processing node. We performed two sets of experiments. The first set, shown in Figure 2 measures the performance on the original MANNA machine. In this set of experiments, we observe that adding the support of I-Structures (*Threaded-C+IS* version) and ISSC (*Threaded-C+ISSC* version), in fact, degrades the system performance. This is because that when the cost to execute split-phase operations is very low (0.875 μs as in MANNA machine), the communication interface overhead saved by caching does not compensate the operation overhead of I-Structures and I-Structure Software Cache.

However, this kind of communication efficiency is usually not available in affordable and widely available networks of workstations. The sending and receiving of network packages may take from hundreds to thousands of cycles depending on the design of the network interface [4]. 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 [13] and Active Messages [14], it still costs 40~60 micro-seconds to send a message to the network.

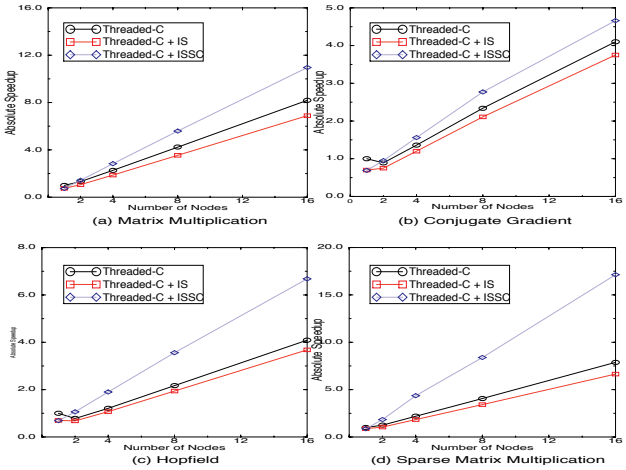

Figure 3. Absolute speedup with 10 μs communication interface overhead

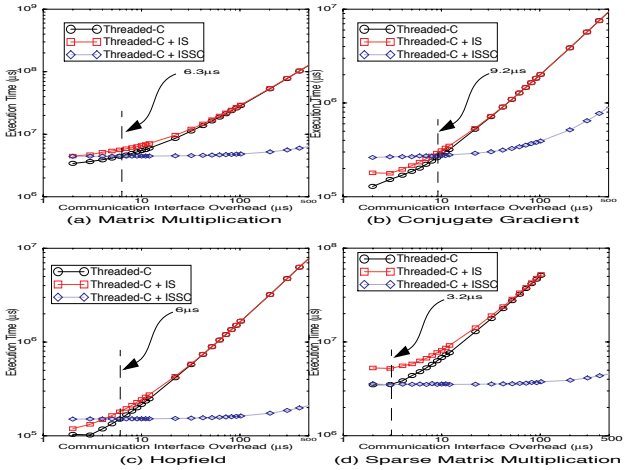

Figure 4. Execution time with synthetically variable communication interface overhead

Thus, in our second set of experiments, shown in Figure 3, we add 10 μs (500 processor cycles) to the communication interface overhead of MANNA machine. In this set of experiments, we observe a significant speedup of the *Threaded-C+ISSC* version over the *plain Threaded-C* and *Threaded-C+IS* versions for all benchmarks. Notice that even with the addition of 10μs, the remote operation cost is far less than the communication interface overhead of fast local area network [13], which cost 40~60 micro-seconds. In this experiment, we show that in spite of the overhead associated with the software implementation of caching scheme for I-Structure, by taking advantage of the global data locality in applications and reducing the number of requests sent in the network, the I-Structure Software Cache makes the system more robust to increased remote cost operations.

From the results presented in Figure 2 and 3, we observe that the communication interface overhead is a determinant factor in the performance of the I-Structure Software Caches. To have a better understanding of the relationship between the ISSC performance and the cost of remote operations, we ran our experiments on a 16 node system with a variable synthetic communication overhead. Figure 4 shows the execution time of applications as the communication overhead increases. For each benchmark, the graph shows the point where the *Threaded-C+ISSC* version starts to out-perform the *plain Threaded-C* implementation. ISSC starts to help the system when the communication interface overhead is greater than 6.3μs, 9.2μs, 6μs and 3.2μs respectively in dense matrix multiplication, conjugate gradient, Hopfield and sparse matrix multiplication. When the communication interface overhead exceeds 100μs, the *Threaded-C+ISSC* versions run almost 10 times faster than the *plain Threaded-C* versions in our benchmarks. The addition of I-Structure and I-Structure Software Cache make fine-grain multi-threading system more profitable in network of workstation environment.

5 Related Work

Dennis and Gao discussed the implementation of caching capabilities for single-assignment structures in shared memory systems [5]. Culler *et al.* [3] implemented software I-Structure caching in the *Id90* compiler for the Threaded Abstract Machine (TAM) implemented on the CM-5. In their implementation, a cache block is formed by a single I-Structure element. Therefore, only temporal data locality could be exploited and no deferred read sharing problem occurs. Our experiments show that exploiting spatial data locality significantly improves performance. Kavi *et al.* [10] proposed a hardware supported cache memories for the Explicit Token Store (ETS) model of data-flow systems with an I-Structure memory system. A write-back cache is adopted in their design. To implement this mechanism, a missing table is maintained in the producer's IS-Cache to indicate the pending status of the I-Structure elements and extra interrogation messages are passed on the network.

6 Concluding Remarks

EARTH is a fine-grain multi-threaded architecture and execution environment. We presented the designs and implementations of I-Structures and I-Structure Software Cache (ISSC) using EARTH Threaded-C language. ISSC caches values obtained through split-phase transactions in the operation of an I-Structure. It also exploits spatial data locality by clustering individual element requests into blocks. Our experimental results show that the inclusion of ISSC in a parallel system that provides split-phase transactions reduces the number of remote memory requests and hence reduces the traffic in the network. The resulting EARTH system is more robust to variations in the latency incurred by remote operations.

In this paper, we demonstrated that even a software implementation of an I-structure cache can yield performance improvements. We also demonstrated that such performance gains will be even more evident in machines with higher latencies, such as network of workstations [9, 13]. The concept of I-Structure caches is not limited to software implementation. While some researchers concentrate on the development of faster network interfaces, I-Structure caches can be implemented with dedicated hardware, a decoupled general purpose processor or even integrated into the design of network interface.

7 Acknowledgements

This research is supported by National Science Foundation under grant award MIP-9707125 and INT-9815742. We acknowledge NSF for the partial support of the current research on EARTH through grants: NSF-CCR-9808522, NSF-CISE-9726388 and NSF-CDA-9703088. We would also like to acknowledge all current and former members of CAPSL who designed, developed, and maintain the EARTH platform and its machines.

References

[1] Arvind, R. S. Nikhil, and K. K. Pingali. I-structures: Data structures for parallel computing. *ACM TOPLAS*, 11(4):598–632, October 1989.

[2] D. H. Bailey, J. T. barton, T. A. Lasinski, and H. D. Simon. The NAS Parallel Benchmarks. Technical Report NASA Technical Memorandum 103863, NASA Ames Research Center, July 1993.

[3] D. E. Culler, S. C. Goldstein, K. E. Schauser, and T. von Eicken. Empirical study of a dataflow language on the CM-5. In G. Gao, L. Bic, and J.-L. Gaudiot, editors, *Advanced Topics in Dataflow Computing and Multithreading*, pages 187–210. IEEE Press, 1994.

[4] 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.

[5] J. B. Dennis and G. R. Gao. On memory models and cache management for shared-memory multiprocessors. ACAPS Technical Memo 90, School of Computer Science, McGill University, Dec. 1994.

[6] G. R. Gao. An Efficient Hybrid Dataflow Architecture Model. *Journal of Parallelism*, 19(4), Dec. 1993.

[7] G. R. Gao, H. H. J. Hum, and Y.-B. Wong. Parallel Function Invocation in a Dynamic Argument-Fetching Dataflow Architecture. In *Proc. of PARBASE-90: Intl. Conf. on Databases, Parallel Architectures, and their Applications, Miami Beach, Florida*, pages 112–116, Mar. 1990.

[8] 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.

[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] K. M. Kavi, A.R. Hurson, P. Patadia, E. Abraham, and P. Shanmugam. Design of Cache Memories For Multi-Threaded Dataflow Architecture. In *ISCA 95*, pages 253–264, 1995.

[11] W. Lin and J.-L. Gaudiot. I-structure Software caches - A split-phase transaction runtime cache system. In *PACT 96*, Oct. 1996.

[12] W. Lin and J.-L. Gaudiot. The Design of An I-Structure Software Cache System. In *MTEAC 98*, Feb. 1998.

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

[14] T. von Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauser. Active messages: a mechanism for integrated communication and computation. In *ISCA 92*, pages 256–266, May 19–21, 1992.

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

[16] 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.