University of Delaware
Department of Electrical and Computer Engineering
Computer Architecture and Parallel Systems Laboratory

A Comparative Performance Study of Fine-Grain Multi-threading on Distributed Memory Machines

Prasad Kakulavarapu†
Christopher J. Morrone
Kevin B. Theobald
José Nelson Amaral
Guang R. Gao

CAPSL Technical Memo 35
November 11, 1999

†School of Computer Science, McGill University, Montréal, Québec, Canada

University of Delaware • 140 Evans Hall • Newark, Delaware 19716 • USA
http://www.capsl.udel.edu • ftp://ftp.capsl.udel.edu • capsladm@capsl.udel.edu

Abstract

We present a comparative study of the implementation of the Efficient Architecture for Running THreads (EARTH) on IBM SP-2, Beowulf, and the MANNA machine. EARTH is a programming, architecture, and execution model that implements fine grain multi-threading. Each platform presents different constraints on the interaction between the EARTH runtime system and the network. Threaded-C, the programming language for EARTH, provides a uniform address space to allow data exchange among the processing nodes in all these distributed-memory platforms. We characterize the performance in each implementation by measuring the cost of EARTH operations, such as the exchange of synchronization signals, the spawning of threads, and the movement of data across processing nodes. Then we will conduct a detailed study of the performance of applications belonging to three different programming models.

Keywords: Fine-grain multi-threading, non-preemptive threads, context-switching, distributed memory, network of workstations.

Contents

List of Figures

List of Tables

1 Introduction

Designing multiprocessor systems that deliver a reasonable price-performance ratio using off-the-shelf processor and compiler technologies is a major challenge. While modern processors can issue multiple instructions per cycle, they lack the features required to address fundamental issues in multiprocessing systems: latency, bandwidth and synchronization overheads. A well designed parallel system must balance the trade-off between a fine task granularity [7] and the impact of communication latencies on performance. Coarse-grain parallel systems can tolerate long latencies if the application provides enough parallelism because each task is long enough to amortize the communication overheads. But coarse grain systems do not fully exploit the parallelism existing in irregular parallelism. Fine-grain parallelism, on the other hand, enables further parallelization of many applications, but has proved to be difficult to support due to the higher relative cost of communication and synchronization latencies [7].

EARTH - Efficient Architecture for Running THreads [2, 8] is a multi-threaded architecture and program execution model that supports fine-grain, non-preemptive, light-weight threads, which are called *fibers* under the model. EARTH is designed to allow the implementation of a multi-threaded execution model with off-the-shelf microprocessors in a distributed memory environment [3]. In order to reduce OS related costs, EARTH fibers operate at the user-level. The EARTH runtime system assumes the responsibility to provide an interface between an explicitly multi-threaded program and a distributed memory hardware platform.

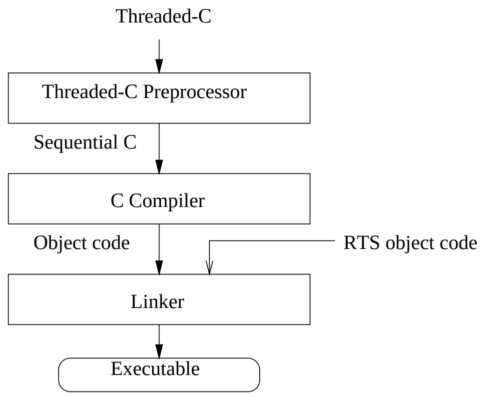

Figure 1: Translation sequence of Threaded-C code

In this paper we present performance results from three implementations of EARTH: EARTH-SP2, EARTH-Beowulf, and EARTH-MANNA. All these implementations run the same application program written in or compiled to Threaded-C, an explicitly multi-threaded extension of C. In all three implementations the Threaded-C code is converted by a pre-processor into ANSI-C with calls to runtime system functions. The translation sequence of Threaded-C programs into the final executable is shown in Fig. 1. The runtime system performs fiber scheduling, context switching between fibers, inter-node communication, inter-fiber synchronization, global memory management, and dynamic load balancing.

Given the EARTH programming and execution model, and its implementation on platforms with different processor-network, processor-memory and network-memory interfaces, we are interested in studying if the EARTH multithreading model can effectively deliver performance improvements for a range of applications across these platforms. One should expect that obtaining performance improvements on tightly coupled architectures should be easier than on loosely coupled ones.

2 EARTH Execution Model

EARTH is a multithreaded program execution and architecture model [2]. Fine-grain, non-preemptive fibers are the atomic units of multithreaded computation in EARTH. A *fiber* is a set of instructions that is executed sequentially in an atomic fashion. Interacting fibers that share context are grouped into bigger units, and are represented in the EARTH runtime system as *tokens*. While the fibers are the atomic units of work, the tokens are the units of parallel work load. The *split-phase* nature of EARTH operations overcome communication latencies by switching to another *ready* (or enabled) fiber, while servicing the communication requirements of another fiber. Applications execute in global memory space comprising the local memories on all nodes in the system. The terms *fiber* and *thread* are used interchangeably in this document.

Application fibers in EARTH, are designed using the *Threaded-C* language, a multithreaded variant of the `C` language. EARTH fibers can be relatively short, typically a few hundred μs on IBM-SP2. The cost for the CPU to perform a context switch is typically only a few tens of machine cycles (about 1μs on SP-2).

3 Dynamic Load Balancing

Scheduling a fiber involves two important decisions: *where* the fiber should be executed, and *when* it should start execution. In the EARTH architecture model, these two decisions are kept separate. It is the responsibility of the load balancer to decide where the tokens will be executed, while the EARTH scheduler decides the local fiber execution order within a node.

There are currently eight load balancers implemented in the EARTH runtime system. The balancers used in this paper are the *dual* and the *his* balancers. The dual balancer is a receiver-initiated balancer, i.e, the load balancing activity is initiated by an idle node, which starts sending requests to destinations decided by the load balancer algorithm. This approach is also called *work stealing*. The his balancer is a hybrid balancer, in which the load balancing activity can be initiated by the same node either as a sender or as a receiver.

The dual load balancer was designed for the dual-processor EARTH-MANNA system. Load balancing information is exchanged in a virtual ring, a subset of the actual interconnection network, each node communicating only with its predecessor and successor. Requests for tokens are passed in anti-clockwise direction, while the tokens themselves are sent in the clockwise direction.

When the current fiber terminates, a token is requested from the local token queue. If no token is available and no request was previously sent, a token request is sent to the predecessor. A node receiving such a request either sends a token back to its successor or sends a token request to its predecessor. The request will travel through all the nodes in sequence until a token is found or a node is reached that has already sent a token request.

It is possible to combine the features of both receiver and sender initiated load balancers. An implementation of this idea is the his (history) balancer which is based on a hybrid algorithm and uses history information.

In the receiver mode, it sends the request directly to the nodes from which a token was last received. In the sender mode, it sends tokens to nodes that are likely to be hungry by using history information in the distribution of spare tokens. This is possible because each node keeps a list of nodes which had sent a token request earlier. This list of hungry nodes is updated after the token is sent away.

4 Hardware Platforms

We select three machines for this comparative study: the MANNA, the IBM SP-2, and the Beowulf. These machines represent different levels of availability, cost, and effort to implement a parallel system. The MANNA is a research machine with dual processor nodes interconnected through a cross-bar switch. The EARTH team had direct access to the network interface and hardware storage in the machine, and thus was able to produce a very efficient implementation of the EARTH model. Only a few installations of MANNA exist. The IBM SP-2 is an inherently parallel machine that is typically available in computing centers. The EARTH team was also granted access to the network card data structures in the IBM SP-2 to enable the EARTH runtime system to directly start network operations. The Beowulf implementation uses exclusively off the shelf components, hardware, network drivers, and operating system. It is the most portable version of EARTH, and the most available because the cost and effort to construct a Beowulf cluster is minimal. However this portability imposes a hit on the latency of the EARTH operations.

The MANNA (Massively parallel Architecture for Non-numerical and Numerical Applications) was developed at GMD-FIRST in Berlin, Germany, in the early 90's [1]. Each node of the machine contains two 50-MHz Intel i860XP RISC processors, each with on-chip data cache and instruction cache of 16KB each. The two processors share 32 MB of DRAM on a common bus, and stay coherent with this memory and each other using bus snooping and the MESI protocol. The bus also runs at 50 MHz. Multiple dual processor nodes of the MANNA are connected through a custom-designed 16×16 packet-switched crossbars. Each input port can accept one data byte per 20ns cycle, and the input is buffered by a FIFO. The crossbar bandwidth is 800 MB/s if all 16 inputs are in use and each transmits to a different output port. The EARTH-MANNA implementation has been described previously [8].

The IBM RS/6000 Scalable POWER Parallel System (SP-2) is a distributed memory multiprocessor. Each processing node is equipped with a 120 MHz POWER2 Super Chip, 128 KB of data cache, 32 KB of instruction cache, at least 64 MB of RAM, and operate with a 256 bit memory bus. The tb-3 switch provides a network interface with a peak hardware bandwidth of 150 MB/s in each direction. A detailed description of the EARTH-SP2 implementation is presented in [5].

The Beowulf cluster [6] is equipped with 200MHz Pentium Pros, each node with 128 MB of RAM. The nodes are interconnected through a 100 Mb/s switched ethernet network. The EARTH inter-node communication and synchronizations are implemented on top of the TCP/IP protocol.

5 Latency of EARTH Operations

The machines that we are studying have different processor and network speeds, and distinct implementations of the EARTH runtime system. The latency of the operations required to communicate and synchronize across processing nodes is a determinant factor in the performance of some applications. In this section we measure the latency of some EARTH operations in all three platforms. These measurements are presented in terms of the number of processor cycles in the machine to facilitate a comparison between the machines. It is important to observe that the processor is not busy with the operation for the number of clock cycles shown in Table 1. Most of the remote operation time is spent either waiting on queues or in the network, releasing the processor to execute other ready fibers.

Table 1 displays the latency of EARTH operations in the three platforms used in this comparative study. In the measurements in the "sequential" column the next EARTH operation is issued after the receipt of a synchronization signal confirming that the current operation is completed. For instance two fibers, fiber 1 and fiber 2, are necessary to measure the latency of a synchronization operation. Fiber 1 issues the operation, and terminates, while the launching of Fiber 2 marks the completion of the operation. After executing the runtime system code for the operation and fiber 2 is enabled, fiber 2 is placed in the ready queue for execution. In the case of the "pipelined" measurements, multiple operations are issued from fiber 1, which then terminates. the elapsed time is measured in fiber 2.

The measurements in the first row of Table 1 are obtained as follows:

Sequential Local: Thread 1 issues a synchronization signal that causes fiber 2 to became enabled. When enabled fiber 2 issues a synchronization signal that causes fiber 1 to became enabled. This cycle is repeated N times (we used $N = 100000$ in our tests). The time required for the N repetitions is measured and the average per synchronization signal is computed.

Sequential Remote: Same as above but fiber 1 and fiber 2 are scheduled in different processors, thus there is a delay of going through the network to perform remote operations.

Machine	Operation	Sequential		Pipelined	
		Local	Remote	Local	Remote
MANNA 1 cycle = 20 ns	Sync Thread	116	199	42.0	49.7
	Spawn Thread	113	213	—	—
	Get_Sync	141	348	56.8	94.0
	Data_Sync	138	333	53.0	90.7
	Fun. Call (1)	250	451	159	140
	Fun. Call (18)	410	628	276	223
SP2 1 cycle = 8.3 ns	Sync Thread	104	2751	24	414
	Spawn Thread	101	2652	—	—
	Get_Sync	122	5366	32.2	699
	Data_Sync	107	5276	27.2	695
	Fun. Call (1)	231	5553	140	784
	Fun. Call (18)	262	5656	171	831
Beowulf 1 cycle = 5.0 ns	Sync Thread	1146	21014	15.7	227552
	Spawn Thread	1193	22863	—	—
	Get_Sync	1211	41614	26.5	11482
	Data_Sync	1201	41513	27.2	37272
	Fun. Call (1)	2416	42728	1228	176389
	Fun. Call (18)	2514	43735	1339	160271

Table 1: Latency of EARTH operations, measured in number of cycles

Pipelined Local: Thread 1 starts the clock and issues N synchronization signals without waiting for any synchronization signal. After receiving N signals fiber 2 is enabled and stops the clock. This version is called "pipelined" because in a machine with separate SU and EU units, the operation of the EU, SU and the network can be superposed in a pipelined fashion. Even in single-processor nodes, this results in performance gains because the sender CPU does not need to wait for a reply from the receiver CPU, before sending the next request. In addition, the synchronization is handled in a different manner with the pipelined version, that results in fewer context-switches than in the sequential style of execution.

Pipelined Remote: Similar, but fiber 1 and fiber 2 execute in different processors. When enabled, fiber 2 sends a synchronization signal to another fiber in the same processor as fiber 1 to stop the clock.

Both in the MANNA and in the IBM-SP2 the EARTH runtime system has direct access to the network interface and can start network operations without any context switching. In fact in the case of the MANNA, the second processor performs all network related operations. In both cases, the runtime system has direct access to the network card data structures which makes network communications and polling faster. This is in contrast to the relatively high

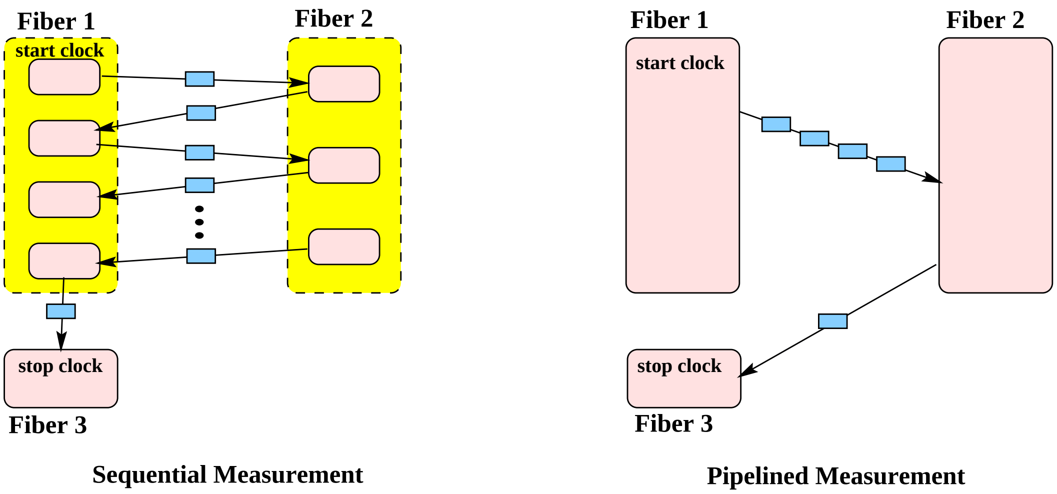

Figure 2: Measuring the latency of a sync operation - (a) Sequential (b) Pipelined

overheads associated with traversing through the TCP/IP stack in the case of the Beowulf. Further, when a message arrives, an interruption is generated to force the operating system to handle the message. This causes a context switching between the EARTH runtime system and the Linux operating system [1]. We are currently reviewing the EARTH-Beowulf implementation to reduce the penalty of the intervening OS actions in the latency of the EARTH operations.

Figure 2 illustrates the sequential and the pipeline measurements. The latency of EARTH operations are shown in Table 1. One observation common to most operations is the high latencies associated with sequential execution when compared to the corresponding pipelined measurements. This is expected, as the overheads associated with issuing the operations sequentially are absent in the pipelined runs. The difference in the processor speeds is very well reflected in the different pipelined speedups for the latencies for local operations (ratio of sequential latencies over pipelined latencies). This ratio is even higher in the case of the Beowulf, because of factors other than the processor speed. The EARTH runtime system polls the network at the termination of every fiber. After responding to synchronization or load balancing requests, execution continues with the next fiber in the ready queue. Since a sequentially issued operation is terminated in another thread, the polling costs add to the local CPU costs.

Remote operations cost less in the MANNA than in the IBM SP-2 or the Beowulf, because of the second processor in the MANNA which takes care of the communication and synchronization operations. The remote costs for sequentially issued operations cost higher in the Beowulf, because of the time required to compose the sending and receiving messages in addition to the polling time.

Pipelined execution of remote operations on the Beowulf is an exception, where the sequential version runs far faster. This is because of the higher context-switching overheads endured

[1]The times reported for the Beowulf runs are “wall clock time” and thus include the costs of the intervening operating system activities. This is a correct measurement because under the current implementation, the user will not be able to distinguish between the time spent in the operating system and in the EARTH runtime system

between the runtime system code, and the operating system code, while sending messages across the network. After executing the runtime system code for the operation, control switches to the operating system to perform the actual communication, after which control again switches back to the runtime system code for issuing the next operation. This switch between the kernel and user space is the reason for the poor performance of remote pipelined operations.

The other EARTH operations measured in Table 1 include the direct spawning of a fiber; a `get_sync` operation in which fiber 1 requests a word of data from fiber 2 and fiber 2 synchronizes fiber 1 when the data arrives; a `data_sync` operation in which fiber 1 sends a word of data to fiber 2 and fiber 2 synchronizes fiber 1 when the data arrives; and function calls with 1 and with 18 parameters, which represent the invocation of a threaded function either in the same node or in a remote node.

6 Performance Measurements

In this section we present performance results for three applications: N-Queens, Paraffins(28), and a dense matrix multiply, in all three platforms. The N-Queens is a typical recursive program that counts how many ways N queens can be placed in an $N \times N$ chess board so that no queen may attack another. In the version that we used, $N = 12$, and the parallelism is "throttled". When four queens are placed on the board, the program switches to a sequential execution and no longer generates migratable tokens. The idea is that at the level of the recursion enough instantiations of the recursive function have been generated to distribute the computation among the processors in the machine.

Paraffins is one of the four "Salishan problems" from the 1988 Salishan High-Speed Computing Conference. Paraffins enumerates all distinct isomers of each paraffin (molecule of the form C_nH_{2n+2}) of size up to a given maximum. The problem solved by paraffins is similar to the problem of detecting isomorphisms in labeled free trees. A list of paraffins is generated and the program returns an array filled with the number of distinct paraffins of each size up to and including the maximum. To exploit parallelism, functions are invoked in all the processors to compute the radicals and then tokens are generated to compute the paraffins of the required size. This benchmark belongs to the irregular class of problems, with irregular communication patterns, and unbalanced computations. In our experiments we measured the performance for Paraffins(28).

The dense matrix multiply algorithm that we used in this study is a simple minded, non-blocking algorithm that computes $C = A \times B$, where A, B and C are $N \times N$ matrices (in our measurements $N = 1024$). Both matrices A and B are stored in node zero and the resulting matrix C is to be also stored in the memory of node zero. Node zero generate migratable tokens that are to compute one row of the matrix C and move the result back to node zero. The first time that a node i executes a token, it copies the entire matrix B to its local memory and the specified row of A. It retains the copy of B to reuse in the computation of future tokens. Although a dense matrix multiply is a very regular algorithm, this version relies in the dynamic load balancer to distributed the load among the processors.

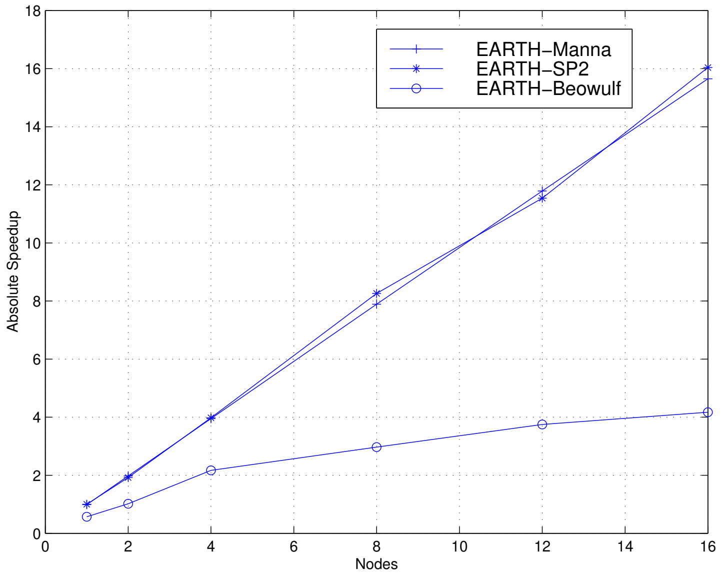

Figure 3: Absolute speedup for Queens(12)

Figures 3, 4, and 5 show the absolute speedups for three benchmarks on each machine. Table 2 displays the actual execution time for the applications in the three platforms. The absolute speedup is measured as the quotient between the time required to execute a sequential version of the code and the time required to execute the parallel version in P processors.

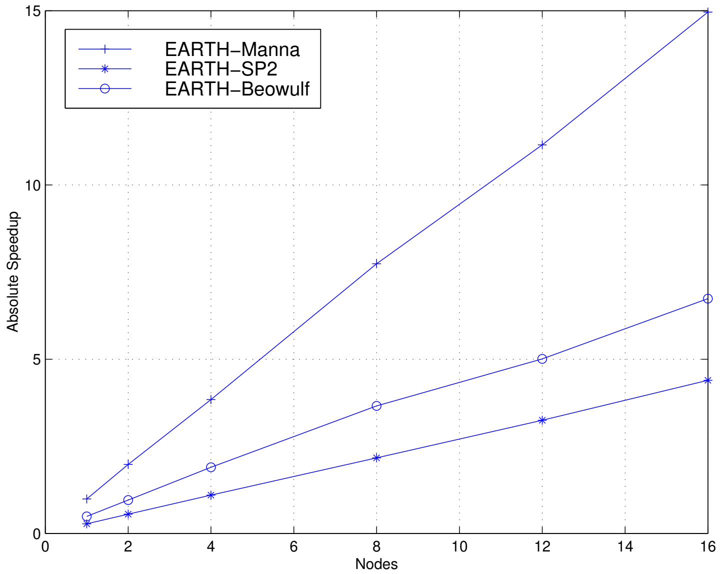

Figure 4: Absolute speedup for Paraffins(28)

An interesting observation to note is the disparity between the CPU speeds and network speeds. In the case of the MANNA, the slow CPU speed results in high elapsed time for sequential execution. In addition, the dual load balancer provides a very simple load balancing algorithm, with minimum overheads. The extra messages generated due to the ring topology adopted in the dual balancer, are compensated by a dedicated processor on each node to deal with the network traffic.

The his balancer on the SP-2 and the Beowulf, on the other hand works on single processor nodes. In order to reduce the network traffic, the his balancer uses history information to send tokens directly to the destination nodes, rather than following the ring topology. This balancer works very well in the case of the IBM SP-2, due to its efficient network interface. However, in the case of the Beowulf, high CPU speed and low network speed result in comparatively poor performance, especially in the case of irregular, and communication intensive applications. Due to the high CPU speed, the computation time is usually not high enough to amortize the remote communication costs.

Another important factor is the *uni-node support efficiency* or USE factor [2, 8]. The USE factor is the ratio of sequential execution time and the elapsed time for one-node parallel execution. An ideal 100% use-factor indicates minimum overheads imposed by the multi-threaded environment, and the presence of enough parallelism in the form of fibers to hide the latencies of the multi-threaded operations. A unity USE factor also symbolizes good absolute speedup, and indicates the possibility of better and equal load balancing on multiple nodes.

In the case of Queens(12), both the MANNA and the SP-2 implementations of EARTH deliver almost linear speedup. The his balancer on the SP-2 performs better than the dual balancer which is tuned for the dual processor MANNA. However the speedup of the Beowulf implementation tapers off after a small number of processors. We believe that this happens mostly because of the iterations between the EARTH runtime system and the Linux operating system actions, including the frequent interruptions to the kernel because of frequent arrival of small messages. In addition, the USE factor on the Beowulf for the Queens benchmark is quite low. This is because of the significant amount of multi-threaded overheads endured, despite the throttling of parallelism.

The IBM SP-2 platform performs best for the irregular application Paraffins (28). However, the elapsed time for sequential execution on the SP-2 is low, resulting in a very low USE factor. This in turn results in poor absolute speedup when compared to the MANNA or the Beowulf. The dynamic computation in this application is handled very well by the high speed processors of the Beowulf when compared to the MANNA. This can be observed in the low sequential execution time for the Beowulf, in contrast to MANNA.

The matrix multiplication application represents the regular class of problems, where the computation time can amortize the minimal multi-threading overheads. This is visible in the near unity USE factors for the SP-2 and the Beowulf platforms. On the other hand, with the MANNA platform, the extremely regular computation requiring lots of memory accesses fails to hide or overlap with the multi-threading overheads on the slow CPU. This application relies

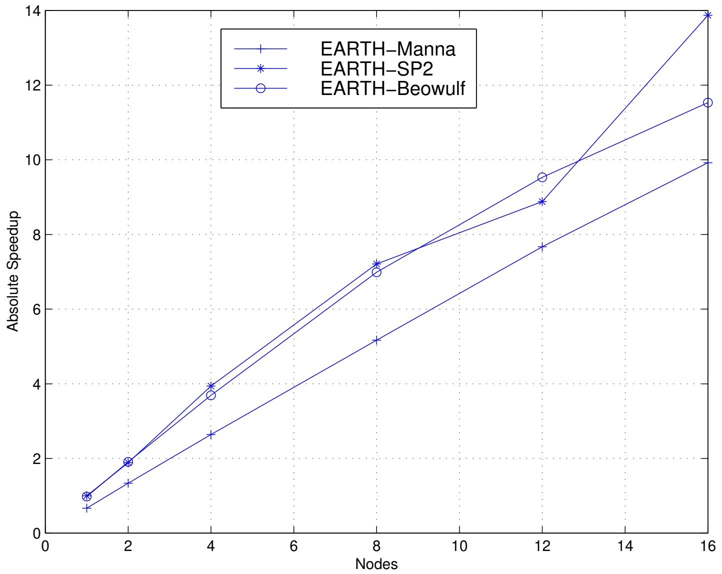

Figure 5: Absolute speedup for Matrix(1024X1024)

Benchmark	Machine	SEQ	Parallel: Num of Processors					
			1	2	4	8	12	16
Queens(12)	MANNA	17.25	17.46	8.74	4.37	2.19	1.46	1.10
	SP2	4.79	4.78	2.50	1.21	0.58	0.41	0.30
	Beowulf	6.63	11.56	6.51	3.60	2.22	1.77	1.59
Paraffins(28)	MANNA	398	398	200	101	51.0	34.7	25.8
	SP2	57.2	205	103	51.8	26.2	17.3	13.1
	Beowulf	168	342	174	88.2	45.9	33.5	24.9
Matrix (1024X1024)	MANNA	364	542	271	138	70.4	36.71	30.70
	SP2	283.47	284	149.95	72	31.97	20.48	39.4
	Beowulf	245	249	128.22	66.27	34.98	25.68	21.22

Table 2: Execution time (in seconds)

a lot on equal distribution of the workload by the load balancer. Here, the his balancer (in the SP-2 and Beowulf) scores very well against the dual balancer in the MANNA runtime system. The long token distribution latencies due to the ring topology, minimal load state information of the dual balancer fail to exploit the regular nature of the application and result in unequal load distribution.

The three machines studied in this paper are quite different. The runtime system is tuned to take advantage of specific features of the MANNA hardware, whereas a portable runtime system is used in the case of the SP-2 and the Beowulf. The Beowulf cluster is the most "off-the-shelf" and most affordable machine; it uses readily available processors, networks, compilers and operating systems. Although commercially available, the IBM-SP2 is not as affordable, and thus is only accessible in computer centers. Further, the runtime system has direct access to the network data structures, which facilitates lower communication overheads when compared to the TCP/IP interface in the Beowulf. Although the MANNA is a very good platform from a computer organization stand-point, it might have a longer execution time than an SP-2 or Beowulf platform for the same number of processors. On the contrary, the Beowulf is a very affordable and interesting option for using off-the-shelf technology; however, the performance for irregular applications is limited by the network speed. The IBM SP-2 performs reasonably well for most applications, with a very effective network interface.

7 Final Remarks

The relatively poor performance for both the EARTH-SP2 and the EARTH-Beowulf for the paraffins benchmark reflects the difference in speed between the processor and the network of these machines. For instance, on Table 1 we observe that a remote sync operation on EARTH-SP2 requires 14 times more cycles than on EARTH-MANNA. On EARTH-Beowulf requires on average 106 more processor cycles to perform a remote sync operation than EARTH-MANNA.

The dense matrix multiplication algorithm used in this study was designed to test the EARTH load balancer. [2] The speedups shown in Figure 2(c) for all three machines demonstrate that the load balancer effectively distributes the processing load among the nodes.

Applications belonging to three different programming models- recursive, irregular and regular classes are studied for their performance on the three different platforms. While the CPU speed, USE factor and the load balancer adopted are seen to affect performance in a major way across all the platforms, the high communication costs associated with the network interface seemed to have a bigger impact on all communication intensive applications in the EARTH-Beowulf.

8 Acknowledgements

We thank Sean Ryan for his valuable suggestions which greatly improved the quality of this paper. We are grateful to GMD First and Cornell Theory Center for allowing us access to their MANNA and IBM SP-2 machines for experiments in this paper. The authors would like to thank current and former members of the ACAPS Lab, McGill University, and the CAPSL Lab, University of Delaware for their insight, ideas and help. The authors also acknowledge the partial support from DARPA, NSA, NSF (under grants NSF-CISE-9726388 and NSF-MIPS-9707125) and NASA. We also acknowledge the support from Defense Advanced Research Projects Agency (DARPA), National Security Agency (NSA) and National Aeronautics and Space Administration (NASA) through a subcontract with Jet Propulsion Laboratory(JPL)/California Institute

[2]Because of data locality, a blocking algorithm would deliver better performance.

of Technology(CalTech) for the Hybrid Technology Multithreaded Architecture Project at University of Delaware. The partial support from DARPA through the DiVA project is also acknowledged. The initial EARTH work was partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] U. Bruening, W. K. Giloi, and W. Schroeder-Preikschat. Latency hiding in message-passing architectures. In *Proceedings of the 8th International Parallel Processing Symposium* [4], pages 704–709.

[2] Herbert H. J. Hum, Olivier Maquelin, Kevin B. Theobald, Xinmin Tian, Xinan Tang, Guang R. Gao, Phil Cupryk, Nasser Elmasri, Laurie J. Hendren, Alberto Jimenez, Shoba Krishnan, Andres Marquez, Shamir Merali, Shashank S. Nemawarkar, Prakash Panangaden, Xun Xue, and Yingchun Zhu. A design study of the EARTH multiprocessor. In Lubomir Bic, Wim Böhm, Paraskevas Evripidou, and Jean-Luc Gaudiot, editors, *Proceedings of the IFIP WG 10.3 Working Conference on Parallel Architectures and Compilation Techniques, PACT '95*, pages 59–68, Limassol, Cyprus, June 27–29, 1995. ACM Press.

[3] Herbert H. J. Hum, Kevin B. Theobald, and Guang R. Gao. Building multithreaded architectures with off-the-shelf microprocessors. In *Proceedings of the 8th International Parallel Processing Symposium* [4], pages 288–294.

[4] IEEE Computer Society. *Proceedings of the 8th International Parallel Processing Symposium*, Cancún, Mexico, April 26–29, 1994.

[5] Prasad Kakulavarapu, Olivier Maquelin, and Guang R. Gao. Design of the runtime system for the Portable Threaded-C language. CAPSL Technical Memo 24, Department of Electrical and Computer Engineering, University of Delaware, Newark, Delaware, July 1998. In ftp://ftp.capsl.udel.edu/pub/doc/memos.

[6] T. L. Sterling, J. Salmon, D. J. Becker, and D. F. Savarese. *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters.* MIT Press, 1999.

[7] Harold S. Stone. *High-Performance Computer Architecture.* Addison-Wesley Publishing Company, 3rd edition, 1993.

[8] Kevin Bryan Theobald. *EARTH: An Efficient Architecture for Running Threads.* PhD thesis, McGill University, Montréal, Québec, May 1999.