

Performance Portability on EARTH: A Case Study across Several Parallel Architectures

Weirong Zhu Yanwei Niu Guang R. Gao

Department of Electrical & Computer Engineering, University of Delaware

{weirong, niu, ggao}@capsl.udel.edu

Abstract

With the rapidly increasing diversity of parallel architectures and the increasing time and labor for developing parallel applications, the performance portability of parallel programs is becoming increasingly important and should be considered when designing parallel execution models, APIs, and runtime system software. This paper analyzes both code portability and performance portability of parallel programs based on the EARTH model – an event-driven fine-grain multi-threaded execution and architecture model. We discuss several design considerations of the EARTH system that contribute to the performance portability of parallel applications. Experiments of four representative benchmarks are conducted on several different parallel architectures, including two clusters listed in the 23rd supercomputer TOP500 list. The results demonstrate that EARTH based programs can achieve robust performance portability across the selected hardware platforms without any code modification or tuning.

1. Introduction

Given recent technological advances in building parallel computing systems, it is now easier and cheaper for engineers and scientists to get access to high aggregate performance parallel systems, such as large clusters, SMP machines, and SMP clusters. On the other hand, with all the computation resources available, current developments of parallel applications often fail to deliver portable performance across different parallel systems. While the design and implementation of parallel applications normally focus on run-time execution performance, experience shows that the time required for developing parallel programs exceeds the execution time by many orders of magnitude. Thus it is very important to ensure that parallel programs are both code portable and performance portable to various parallel machines.

Another term for *code portability* is *compiling portability* [16]. It means that the code of parallel applications targeted to one platform can also be recompiled and executed correctly when ported to different hardware platforms. This is generally true for parallel programming tools available today.

However, adequate *performance portability* to various platforms is seldom achieved. *Performance portability* requires that a parallel program achieves good performance on all the platforms without code modification. More specifically, *performance portability* tries to address the following problem: “If one parallel implementation of one application already achieves good performance on one platform, can this implementation also achieve same or similar performance on other platforms?” It is obvious that the code portability is the *prerequisite* of *performance portability*.

Application developers are the customers of parallel programming tools and definitely wish to achieve both code portability and performance portability of the developed parallel programs based on the following reasons. Firstly, we hope that upgrading to new hardware can increase application performance proportional to the increase hardware performance. Secondly, we hope that the same parallel program can achieve the same or similar performance on all types of widely available supercomputers, such as uni-processor clusters, SMP machines and SMP clusters. Thirdly, we hope that the performance of applications developed and tuned with a small number of processors can scale well with a large number of processors.

One reason why it is difficult to achieve performance portability is that there is usually a performance tuning phase for the development of parallel programs [8]. During the costly tuning phase, parallel programs are always optimized for specific platforms via timing profiling and feedback. This normally results in good performance on the targeted platform; however, the performance on other platforms may not be so good due to the diversity of parallel architectures. That is, performance portability can not always be guaranteed.

EARTH (Efficient Architecture for Running THreads)

execution model [21] is an adaptive, robust, event-driven multi-threaded execution model with dynamic load balancing support mechanism. The current implementation of EARTH runs on top of parallel machines built with off-the-shelf microprocessors, and takes advantage of fine-grain multithreading to efficiently overlap communication and computation. Experience [22, 10, 25, 6] shows that parallel programs based on the EARTH execution model can achieve good performance on various platforms.

This paper is the first attempt to analyze both *code portability* and the *performance portability* of the EARTH model. We will demonstrate that EARTH programs are portable in the sense of both code and performance portability with the support of a newly developed EARTH Runtime System (RTS) 2.5 across several widely used parallel architectures – uniprocessor clusters, SMP machines, and SMP clusters. The rest of this paper is organized as follows. In Section 2, the EARTH model and runtime system are briefly introduced. In Section 3, we analyze how EARTH based parallel programs achieve both code portability and performance portability. Section 4 shows performance portability of four representative benchmarks on EARTH. Related work is presented in Section 5, followed by a conclusion and future works in Section 6.

2. EARTH model

EARTH [21] is an event-driven multi-threaded architecture and execution model. An EARTH computer consists of a set of EARTH nodes connected by a communications network. As shown in fig 1, each EARTH node consists of an Execution Unit (EU) and a Synchronization Unit (SU) linked to each other by queues. The EU executes active fibers, and the SU handles the synchronization, scheduling of fibers and communication with remote processors.

To assure flexibility and portability, the EARTH model does not specify a particular instruction set. Instead, ordinary arithmetic and memory operations use whatever instructions are native to the processor(s) serving as the EU. The EARTH model specifies a set of EARTH operations for synchronization and communication. These operations are mapped to native EU instructions. To maximize portability, the EARTH model makes minimal assumptions about memory addressing and sharing.

The EARTH execution model defines a two-level execution hierarchy: threaded procedures and fibers. A fiber is a *sequentially executed, non-preemptive, atomically-scheduled* set of instructions. Interacting fibers sharing context are grouped into a bigger unit – threaded procedure. The parallelism of the EARTH execution model is realized through asynchronous function calls of threaded procedures. Invoking a threaded procedure results in the concurrent execution of caller and callee. Execution of a mul-

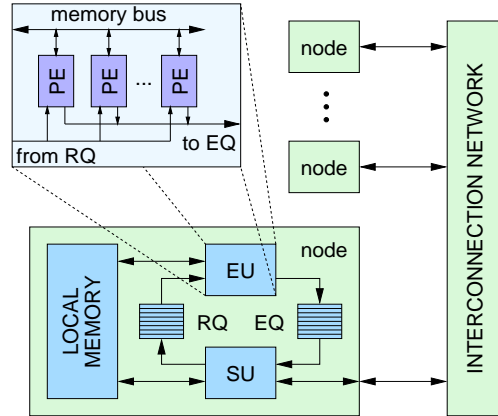


Figure 1: The EARTH architectures

ti-threaded program with concurrent threaded-procedures invocations leads to dynamic unfolding of the computation [11].

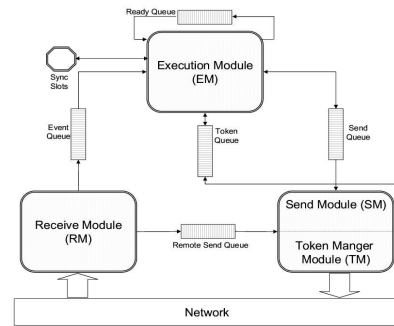


Figure 2: Layout of EARTH RTS 2.5

The EARTH architecture executes applications coded in Threaded-C [23], a multithreaded extension of ANSI-C programming language. By incorporating EARTH operations, Threaded-C can allow the programmer to explicitly indicate the parallelism. Threaded-C is a complete programming language in the sense that it provides constructs that fully capture the multithreading model and gives the programmer complete control of thread construction and thread launching.

The current implementation of the EARTH Execution Model is EARTH RTS 2.5 [18, 14]. The EARTH RTS assumes the responsibility to provide an interface between an explicitly multithreaded program and the underlying hardware platform, which could be either a distributed memory or a shared memory system. The RTS performs thread scheduling, inter-node communication, inter-thread synchronization, global memory management, and dynamic load balancing. The design

of RTS 2.5 is shown in Fig.2. RTS runs on each machine which could be either a physical node within a cluster or a whole SMP machine, each of which is implemented by one POSIX thread. The RTS can be divided into several modules. One or more *Execution Modules* (EM) are responsible for executing the Threaded-C application code. The *Sender Module* (SM) and the *Receive Module* (RM) perform all of the networking operations. A *Token Manager Module* is dedicated to the dynamic load balancing. All modules communicate through the queues shown in Fig. 2.

3. Portability of EARTH program

From our experiences, once a parallel program based on the EARTH program execution model (PXM) is developed and tuned on one specific platform with notable performance, it is expected that both *code portability* and *performance portability* can be achieved on various platforms. In this section, we will analyze how code portability across various platforms is achieved for EARTH programs. Then, we will explain several design considerations of the EARTH Model and the runtime system, which guarantee the performance portability of EARTH based parallel programs.

3.1. Code portability of EARTH programs

The code portability of an EARTH program is ensured by several features of the EARTH Virtual Machine [21], which is defined based on the EARTH PXM and is the basis on which the EARTH RTS is implemented. It specifies the interface between the programmer/compiler and the hardware. The goal of the EVM is “to define a common set of operations in sufficient detail to permit both multiple hardware platforms and multiple high-level languages to be designed around this set” [21].

3.1.1. EVM memory address space model The design of the EVM-A (an EVM based on *Addresses*) is based on the global address space because the most intuitive way to reference memory locations globally is to extend the concept of the local memory address to the global address.

For cluster machines, memory is not physically shared by all the computing nodes. Accordingly, in the EVM-A model, each node has its own memory space, which is divided into two parts: *replicated address space* and *non-replicated address space*. The replicated space must be identical on all nodes. A copy of the executable used by the EARTH program is placed at the same address (base address) in the replicated space on each physical node. This ensures that any identical addresses (branches, sequential function calls, threaded procedures, fibers and static data structures) in the executable will refer to the identical code or data structure on every node. The dynamically allocated

objects, stacks, and frames which hold the contexts of the instances of the threaded procedures are put in the non-replicated space. The global address is defined as the combination of the node id and the local address on that node in order to make the memory location in the non-replicated space accessible by a remote node. EVM-A requires all global address accesses to be handled by defined EARTH operators, instead of direct load and store instructions.

For SMP machines, the memory space is shared by all processors. There is only one copy of the executable in this space. Theoretically, the processor is able to access any memory location in the memory space. However the EVM-A still requires the memory accesses to other threaded procedures’ frames and heaps to be handled by the defined EARTH operators in order to ensure portability.

Therefore using EARTH operators implemented properly in RTS to handle the memory address is an important feature to ensure portability on both clusters and SMP machines. In the EARTH RTS, a “node id” in Threaded-C is a unique identifier of an Execution Module (EM) in the RTS. When a global address is passed to RTS, the RTS extracts both the identity of the node and the local address on that node. The RTS will then determine whether the node is local or remote and perform all necessary corresponding operations.

In summary, the EARTH model provides application developers a consistent way of global address accessing on different platforms. The address space is smoothly extended from a single-CPU machine to a cluster and from an SMP machine to an SMP cluster, thus making the EARTH program code portable to various platforms. The Threaded-C [23] programmers do not need to know the underlying platform. Instead, they only need to know that a number of nodes are available for running the threaded procedures and fibers.

3.1.2. EVM-A thread model The EVM-A thread model is defined based on the EVM-A memory model. Design issues of the thread model include threaded procedure invocation and thread synchronization. The code for a threaded procedure is stored at the same address in the replicated space of each physical node, which means the procedure can be invoked on a remote machine by a globally uniformed reference. In addition, specific EARTH operators are used to increase or decrease the count of the sync slots for a threaded procedure synchronizing with another threaded procedure. A reference to another procedure’s sync slots is also through global address, which will be transparently handled by the RTS.

It can be seen that threaded procedure invocation, and inter procedure synchronization can be coded in one consistent way, no matter what underlying platforms are used. The RTS is responsible for the correctness of all those EARTH operations on different platforms. Differences among vari-

ous platforms are handled by the EVM-A thread model and thus hidden from the users.

In summary, the code portability of EARTH is mainly guaranteed by the successfully defined EARTH EVM-A and the corresponding implementation of the EARTH RTS.

3.2. Performance portability of EARTH program

Once code portability is guaranteed, the next issue is whether the performance is portable to various platforms. We try to address this issue from the perspective of several implementation considerations of the EARTH EVM – the EARTH RTS.

In this section, we will first review the main features of the RTS itself and demonstrate that it can handle communication and computation efficiently by overlapping communication and computation whenever possible. Then, we will discuss two types of applications: low communication applications and communication intensive applications. Finally we will explain why they are portable to various platforms.

3.2.1. Communication and computation of RTS The design of the EARTH RTS 2.5 is tailored to achieve portable performance on various platforms – uni-processor clusters, single SMP machines, and SMP clusters.

In the RTS, the computations are handled by the EM. Each EM corresponds to one conceptual node in a Threaded-C program. The RTS is responsible for creating EMs on different machines, which is transparent to the programmer. EMs are fed with EVM threaded procedures and fibers to be kept busy. When running on SMP nodes, multiple EMs are allowed to be created. Keeping EMs usefully busy is, indeed, the key feature of the event-driven multithreaded EARTH PXM, which has led to robust performance as long as there is plenty of parallelism. It is worth noting that the parallelism of the application can be explicitly specified by the Threaded-C program. In RTS 2.5, the EM will never yield the CPU by itself; instead, it always tries to find and execute new enabled fibers from its ready queue. Thus EM can quickly respond to the new ready fibers although it may also perform some EARTH operations.

The communications are handled by the RM and the SM. All remote procedure invocations, remote memory accesses, and remote synchronizations requested by the EMs are handled by the RM and SM. When no such requests exist, the RM and SM yield CPU resources. Therefore, EMs can get better chance to occupy the CPU as much as possible. When network a I/O requests happen, the RM and SM will be woken up by the OS. They will handle the requests immediately and go to sleep again. So, the computation and communication within one node is well overlapped.

In conclusion, the available parallelism represented by threaded procedures and fibers keep the EMs as busy as possible, while communication requests are quickly responded to by the network modules. The RTS tolerates a relatively large range of communication latency as long as the communicated data can arrive in a timely manner to keep the EM busy. This is the fundamental reason that the Threaded-C program can work similarly well on various platforms.

3.2.2. Performance portability of two types of application programs A high computation and low communication application coded with Threaded-C has good performance portability due to the following reasons. From the fact that satisfying performance is already shown on one platform, it can be concluded that this particular Threaded-C program is well coded and there is enough parallelism available. Because the amount of communication is relatively small compared to computation in this type of application, there is no significant difference when running on different platforms.

For a communication intensive application, achieving scalable performance is a big challenge for any parallel program execution model. Normally, a lot of coding/profiling/tuning work is involved to achieve an efficient implementation. The performance of an EARTH based application is portable from cluster to SMP machines due to the following reasons.

Firstly, since EARTH PXM is an event-driven, dataflow-like programming model and the Threaded-C program can be described by a dataflow-like graph, the EARTH EVM has the same inter-procedure communication patterns for all platforms.

Secondly, with the same communication pattern, the main difference among various platforms is the communication latency. For a cluster machines, the communication is done by a network and normally has long latencies, while for SMP machines, the communication is done through direct memory transactions with smaller latencies. As stated previously, the EARTH RTS can tolerate a relatively large range of communication latencies and maintain good performance. Also the Threaded-C programmer can explicitly overlap communication and computation by making use of the two level thread hierarchy of the EARTH PXM. Therefore, the latency difference across platforms will not have much impact on the performance of the well developed EARTH based programs. The CG [6] implementation is one such example.

Thirdly, the Threaded-C program does not use mutual exclusion to perform synchronizations since excessive mutual exclusions are usually the performance killers on parallel applications for other parallel APIs targeted specifically for SMP. Therefore, the performance of a communication intensive EARTH program is portable from a cluster to an SMP machine. In the future, by integrating state-of-the-art

Table 1: Experiment platforms

Name	Location	Machine Type	Processor type	# of CPUs	Memory per node	Network
Earthquake	UDel ^a	cluster	PIII 500MHz	16 × 1 per node	256M	100T Ethernet
Comet	UDel	SMP cluster	AMD Athlon 1.4G	18 × 2 per node	512M	100T Ethernet
Chiba City [3]	ANL ^b	SMP cluster	PIII 500MHz	256 × 2 per node	512M	Gigabit Ethernet
JAZZ [2]	ANL	cluster	Xeon 2.4GHz	175 × 1 per node	2G	Gigabit Ethernet
Biowolf	UDel	SMP cluster	Xeon 2.8GHz	128 × 2 per node	2G	Gigabit Ethernet
DNA-RNA1	UDel	Sunfire 4800 SMP	Sparc 750MHz	12	24G (total)	N/A

^a UDel = University of Delaware

^b ANL = Argonne National Laboratory

high performance networking to the EARTH RTS, we can expect that the performance is also portable from SMP machines to clusters.

3.2.3. Portability of dynamic load balancing One important feature and advantage of EARTH PXM is the support of dynamic load balancing. In the case of dynamic load balancing, the work load balancing is determined at runtime by the RTS, the same load balancing algorithm is used for all platforms. Therefore the performance is still portable when the communication latency is small enough. To implement an efficient and performance portable dynamic load balancing algorithm in RTS itself is an important research topic.

Up to this point, we can see that one of the great properties of the EARTH Execution Model is that the same program can be portable to various platforms without any modification and still maintain the same or similar performance.

4. Experiments

In this section, we will first introduce a new metric of the performance portability and the detailed specifications of the experimental platforms. Then we will report the performance portability results, observations, and detailed analysis for four representative benchmarks.

4.1. Metric of performance portability

Normally, a parallel program is developed and the performance is tuned based on a specific system or platform, which we denote as the *base system*. Programs with good code portability can be ported without code modification and further performance tuning to a new system, which we denote as the *target system*. Given a parallel program \mathcal{P} developed on the *base system*, let S_n^B denote the absolute speedup achieved by \mathcal{P} on n computing nodes of the *base system*, and S_n^T denote the absolute speedup achieved by \mathcal{P} on n computing nodes of the *target system*. Define the *performance portability* ($P_n^{\mathcal{P}}(b \rightarrow t)$) on n computing nodes for porting the program \mathcal{P} from the *base system* to the *tar-*

get system as:

$$P_n^{\mathcal{P}}(b \rightarrow t) = \frac{S_n^T}{S_n^B} \times 100\% \quad (1)$$

To simplify the notation, we use P_n to denote $P_n^{\mathcal{P}}(b \rightarrow t)$, knowing that the program \mathcal{P} , the *base system* and the *target system* can be easily figured out from the context. In the following subsections, we will use this new metric to evaluate the performance portability of EARTH programs.

4.2. Computational platforms

Experiment platforms are listed in Table 1. The JAZZ cluster [2] in Argonne National Laboratory is ranked 235 among the 23rd TOP500 [1] list, while the Biowolf in Univ. of Delaware is ranked 452.

4.3. Performance portability of EARTH-based programs

Four representative benchmarks are used to verify the performance portability of parallel programs coded by Threaded-C targeted to EARTH.

4.3.1. Traveling salesman problem The Traveling Salesman Problem (TSP) is a graph-theoretic problem. It tries to find the Hamiltonian cycle with the least weight in a weighted graph. The implementation of TSP on EARTH uses a divide-and-conquer approach to enumerate all the possible routes and returns the one with the least weight. All the recursive function calls are generated as TOKENs, which are distributed at runtime to different processors by the dynamic load balancer of the EARTH RTS and executed in parallel.

In our experiment, the base system for developing the parallel TSP is the Earthquake. Figure 3(a) plots the performance portability curve of the EARTH implementation of TSP on different platforms. When the program is ported from a uni-processor cluster (Earthquake) to an SMP cluster (Comet), and an SMP machine (DNA-RNA1), performance portability reaches more than 100% for most cases,

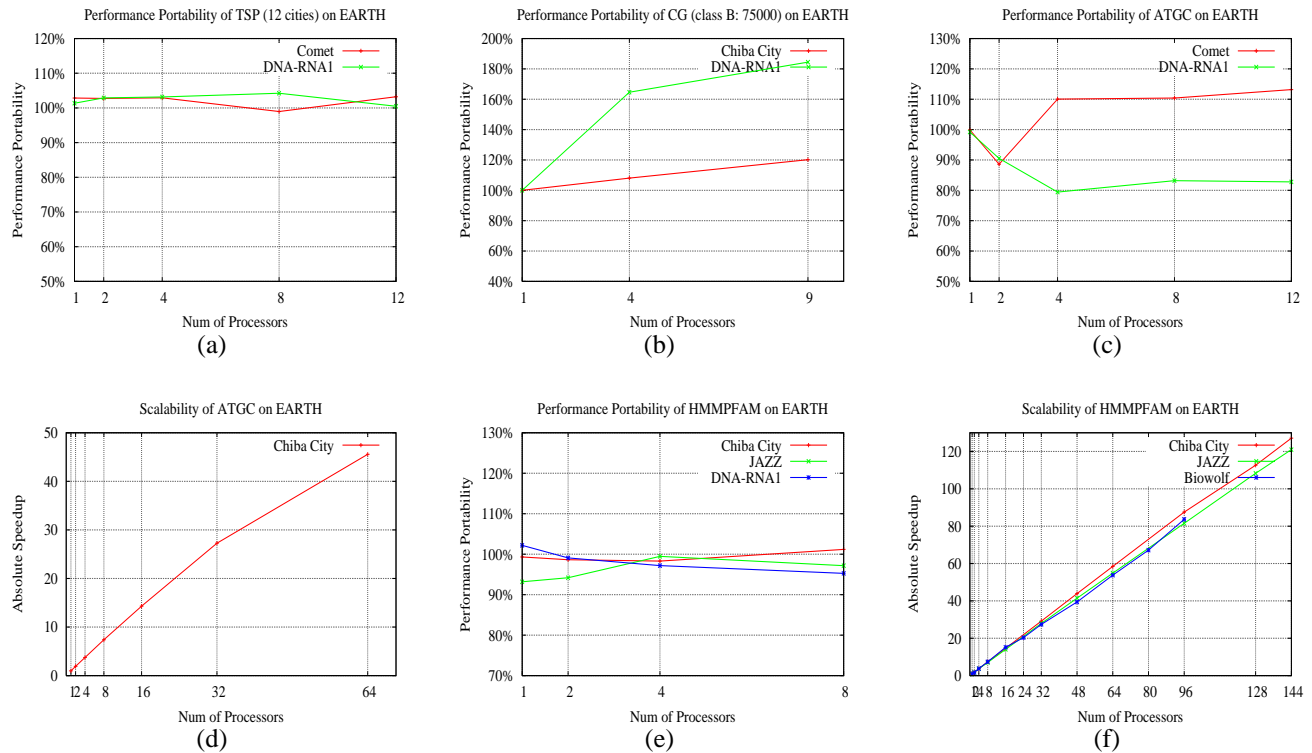


Figure 3: Performance portability of (a) TSP; (b) CG; (c) ATGC; (d) Scalability of ATGC; (e) Performance portability of HMMPFAM; (f) Scalability of HMMPFAM on large-scale clusters

notwithstanding the diversity of underlying hardware platforms. The TSP benchmark demonstrates that the performance of an EARTH based program with *dynamic load balancing* is portable to various parallel architectures.

4.3.2. Conjugate gradient problem Conjugate Gradient (CG) is one of the most widely used iterative algorithms to solve large sparse linear systems of equations. The CG program is both computation and communication intensive. The EARTH based implementation of CG [6] tries to reach the balances between the inter-phase and intra-phase communication costs and minimizes the overall communication costs by using a two-dimensional blocking method. Taking advantage of EARTH's adaptive, event-driven multi-threaded execution model, communication and computation can be well overlapped.

In our experiment, the base system for developing CG on EARTH is the Comet. Scalability results of CG on a large cluster are demonstrated in [6].

Two performance portability curves are plotted in Fig. 3(b) for porting CG of problem size B (75000) to another cluster (Chiba City) and an SMP machine (DNA-RNA1).

There are several observations: first of all, the EARTH implementation of CG is originally targeted to clusters (for instance, Comet), which explains why almost 100% perfor-

mance portability is achieved when the program is ported from Comet to Chiba City. Secondly, super-linear speedup is achieved on an SMP machine DNA-RNA1, and the performance portability is more than 150% on 4 or 9 CPUs. The reason is that the execution of CG will generate large amounts of memory transactions/network communications. With the increase of number of processor, the available memory bandwidth is increased proportionally on parallel machines, which will result in the super linear speedup. However, when running on a cluster, the long latency of the network somehow cancels out this advantage. Another advantage of the DNA-RNA1 is that its Ultrasparc III processor, has much larger L1 and L2 caches than the Pentium III processor of Chiba City cluster and Athlon processor of COMET cluster, thus CG has much better cache performance on DNA-RNA1 than on clusters. The results in [6] also show that the EARTH implementation of CG can achieve scalable and portable performance on a large number of computation nodes. For example, the EARTH implementation of CG achieves a speedup of 41 on 64 nodes on Chiba City for problem size C (150000), while the NAS MPI-based version [5] only gets 13. Therefore, we can see that the communication intensive applications on EARTH, such as CG, can also maintain performance portability from a cluster to an SMP machine.

4.3.3. Another tool for genome comparison Another Tool for Genome Comparison (ATGC) [10] is a biological sequence alignment tool using the dynamic programming algorithm. The multithreaded implementation based on EARTH constructs the similarity matrix from input sequences, and then divides the matrix into strips and further divided into rectangular blocks. Each strip is assigned to a computing node. In order to overlap communication and computation, blocks on a strip are calculated in parallel by EARTH fibers.

The base system for developing and tuning ATGC is the Earthquake, on which the ATGC EARTH implementation can achieve a speedup of 11 with 12 CPUs. Figure 3(c) shows that acceptable performance portability is also achieved on Comet (more than 110%) and DNA-RNA1 (more than 80%) when the program is ported from Earthquake. One reason for ATGC to achieve better performance on COMET is that Comet has the largest *CPU rate / network latency* ratio among three platforms. Fig. 3(d) shows that the EARTH based parallel implementation of ATGC using the EARTH Execution model scales very well and that performance of ATGC is portable when running on a large number of computation nodes. The execution time is reduced from 324 seconds to 7 seconds with 64 nodes.

4.3.4. Parallel HMMPFAM HMMPFAM [4] is a widely used tool for searching a single sequence against an HMM database. The EARTH implementation [25] of HMMPFAM using the dynamic load balancing approach is firstly designed as a cluster based solution, which demonstrates near linear speedup on Chiba City and JAZZ with more than two hundred processors.

We originally developed and tuned the parallel HMMPFAM on Comet, which is the base system to evaluate the performance portability of HMMPFAM on EARTH. Figure 3(e) shows the performance portability results of the HMMPFAM based on EARTH. We can see that programs originally developed on a cluster also has good performance on SMP machines. We also compare the speedup curves achieved by Hmmpfam on three large-scale clusters: Chiba City, JAZZ and Biowolf. The JAZZ is a uni-processor cluster, while the Chiba City and Biowolf are SMP clusters. Fig. 3(f) shows that even executed with a large number of processors, the program is still performance portable to supercomputers with diverse organizations.

4.3.5. Main observations The experiment results of four representative benchmarks demonstrate the performance portability of different kinds of applications on EARTH: 1) computation intensive applications, such as TSP, ATGC, and HMMPFAM; 2) communication intensive applications, such as CG; 3) applications with load balancing challenges, such as TSP, and HMMPFAM; 4) embarrassingly parallel applications, such as HMMPFAM.

All of them achieved greater than 80% performance portability on various parallel machines, which proves that parallel programs based on the EARTH model have performance portability across various hardware platforms without any further code modification and tuning.

5. Related works

A number of other projects have addressed the code and performance portability of other popular parallel APIs (*e.g.*, MPI (Message-Passing Interface), PVM (Parallel Virtual Machine) and OpenMP).

MPI has become an industrial standard for developing parallel applications, especially on clusters. Recently, with the prevalence of the SMP clusters, a lot of work has been conducted to improve MPI to support SMP clusters in order to get the portability for current parallel applications implemented by MPI: 1) the MPICH [9] developed by Argonne National Laboratory 2) Hong Tang and Tao Yang's work – TMPI [20], and 3) MPICH-PM/CLUMP for SMP clusters by Tokahashi and his colleagues [19].

PVM is another widely used parallel API for clusters. Various groups have taken efforts to achieve efficient PVM implementation on SMP machines. These works generate different variations of PVM including 1) PM-PVM (Portable Multithreaded PVM) [17], 2) LPVM (lightweight-process PVM) [24], and 3) TPVM [7].

OpenMP is the most widely acceptable API for parallelizing applications on multiprocessors. There are many efforts to make OpenMP programs portable to platforms other than shared memory systems: 1) the first OpenMP system implementation on a network of SMPs by Lu *et al.* [13], 2) ParADE [12], an OpenMP programming environment for a cluster of SMPs on top of the multithreaded, software distributed, shared memory system (SW-DSM), and 3) "cluster-enabled" OpenMP compiler by Ojima *et al.* [15].

6. Conclusions and future work

This paper analyzes code portability and performance portability of parallel programs based on the EARTH execution model. We concluded that the code and performance portability of EARTH programs is ensured with the event-driven, fine-grain, multithreaded execution model, the design of the EARTH virtual machine, and the efficient implementation of the EARTH RTS. Four representative EARTH based applications are executed on various platforms, including two clusters listed in the 23rd supercomputer TOP500 list. The experimental results demonstrated that the performance of EARTH programs are portable on various parallel platforms. This paper opens an interesting topic for future research and development on parallel programming tools. In the future, we will port more applica-

tions, especially irregular to the EARTH model and examine their performance portability via detailed profiling on different platforms.

7. Acknowledgments

This research is funded in part by NSF, under the NGS grant 0103723, DOE, grant number DE-FC02-01ER25503. We thank other US Federal agencies and the State of Delaware for their support. We thankfully acknowledge the experiment platform support from the HPC Systems Group and the Laboratory Computing Resource Center at Argonne National Laboratory, and the Delaware Biotechnology Institute. We also acknowledge all previous related works by the CAPSL members in the EARTH project, especially Dr. Kevin B. Theobald, who defined the EARTH Execution Model and EARTH Threaded Virtual Machine in his PhD dissertation, Chuan Shen, who co-developed the EARTH RTS 2.5, Fei Chen, who developed the CG on EARTH, and Juan del Cuvillo, who developed the ATGC. We also thank useful discussions from other members of the CAPSL group, in particular Geoff Gerfin.

References

- [1] The 23rd top500 supercomputer list for june 2004. <http://www.top500.org/list/2004/06>.
- [2] The Argonne JAZZ cluster, laboratory computing resource center (lrc). <http://www.lrc.anl.gov/jazz/>.
- [3] The Argonne scalable cluster. <http://www-unix.mcs.anl.gov/chiba/>.
- [4] HMMER: sequence analysis using profile hidden Markov models. <http://hmmer.wustl.edu/>.
- [5] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0.
- [6] F. Chen, K. B. Theobald, and G. R. Gao. Implementing parallel conjugate gradient on the EARTH multithreaded architecture. In *IEEE International Conference on Cluster Computing (CLUSTER'04)*, San Diego, California, Sept 20-23 2004. Accepted, to appear.
- [7] A. Ferrari and V. Sunderam. Multiparadigm Distributed Computing with TPVM. *Concurrency - Practice and Experience*, 10(3):199–228, Mar. 1998.
- [8] I. Foster. *Designing and building parallel programs: concepts and tools for parallel software engineering*. Addison-Wesley, Reading, MA, USA, 1995.
- [9] W. D. Gropp and E. Lusk. *User's Guide for MPICH, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.
- [10] G. G. Juan del Cuvillo, Xinmin Tian and M. Girkar. Performance study of a whole genome comparison tool on a hyper-threading multiprocessor. In *Fifth International Symposium on High Performance Computing*, Tokyo, Japan, October 2003.
- [11] P. Kakulavarapu, O. Maquelin, and G. R. Gao. Design of the runtime system for the portable threaded-c language. *CAPSL Technical Memo 24*, 1998.
- [12] Y.-S. Kee, J.-S. Kim, and S. Ha. ParADE: An OpenMP programming environment for SMP cluster systems. In ACM, editor, *SC2003: Igniting Innovation*. Phoenix, AZ, November 15–21, 2003.
- [13] H. Lu, Y. C. Hu, and W. Zwaenepoel. OpenMP on network of workstations. In *Proc. of Supercomputing'98*, Oct. 1998.
- [14] C. J. Morrone. An EARTH runtime system for multiprocessor/multi-node Beowulf clusters. Master's thesis, Univ. of Delaware, Newark, DE, May 2001.
- [15] Y. Ojima, M. Sato, H. Harada, and Y. Ishikawa. Performance of cluster-enabled OpenMP for the SCASH software distributed shared memory system. In *Proc. of the 3rd IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGrid'03)*, pages 450–456, May 2003.
- [16] R. Reussner and G. Hunzelmann. Achieving performance portability with SKaMPI for high-performance MPI programs. In *ICCS '01: Proceedings of the International Conference on Computational Science-Part II*, pages 841–850. Springer-Verlag, 2001.
- [17] C. Santos and J. Aude. PM-PVM: A portable multithreaded PVM. In *13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, San Juan, Puerto Rico, April 12-16 1999.
- [18] C. Shen. A portable runtime system and its derivation for the hardware SU implementation. Master's thesis, Univ. of Delaware, Newark, DE, December 2003.
- [19] T. Takahashi, F. O'Carroll, H. Tezuka, A. Hori, S. Sumimoto, H. Harada, Y. Ishikawa, and P. H. Beckman. Implementation and evaluation of MPI on an SMP cluster. In *Proceedings of the 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, pages 1178–1192. Springer-Verlag, 1999.
- [20] H. Tang and T. Yang. Optimizing threaded MPI execution on SMP clusters. In *Proceedings of the 15th ACM International Conference on Supercomputing (ICS-01)*, pages 381–392, New York, June 17–21 2001. ACM Press.
- [21] K. B. Theobald. *EARTH: An Efficient Architecture for Running Threads*. PhD thesis, May 1999.
- [22] K. B. Theobald, G. Agrawal, R. Kumar, G. Heber, G. R. Gao, P. Stodghill, and K. Pingali. Landing CG on EARTH: A case study of fine-grained multithreading on an evolutionary path. In *Proceedings of Supercomputing'2000*, Dallas, TX, Nov. 2000. IEEE and ACM SIGARCH.
- [23] G. Tremblay, K. B. Theobald, C. J. Morrone, M. D. Butala, J. N. Amaral, and G. R. Gao. Threaded-C language reference manual (release 2.0). *CAPSL Technical Memo 39*, 2000.
- [24] H. Zhou and A. Geist. LPVM: a step towards multithread PVM. *Concurrency: Practice and Experience*, 10(5):407–416, Apr. 25 1998.
- [25] W. Zhu, Y. Niu, J. Lu, C. Shen, and G. R. Gao. A cluster-based solution for high performance hmmpfam using earth execution model. In *IEEE International Conference on Cluster Computing (CLUSTER'03)*, pages 30–37, Hong Kong, P.R. China, December 2003.