

A Cluster-Based Solution for High Performance Hmmpfam Using EARTH Execution Model

Weirong Zhu Yanwei Niu Jizhu Lu Chuan Shen Guang R. Gao
Department of Electrical & Computer Engineering, University of Delaware
Newark, Delaware 19716, U.S.A
{weirong, niu, jlu, chuan, ggao}@capsl.udel.edu

Abstract

Hmmpfam is a widely used computation-intensive bioinformatics software for sequence classification. The contribution of this paper is the first largely scalable and robust cluster-based solution of parallel hmmpfam based on EARTH (Efficient Architecture for Running Threads), which is an event-driven fine-grain multi-threaded programming execution model. Compared with the original PVM implementation, our implementation shows notable improvements on absolute speed-up and better scalability. Experiments on two advanced supercomputing clusters at Argonne National Laboratory (ANL) achieve an absolute speedup of 222.8 on 128 dual-CPU nodes for a representative data set, which means that the total execution time is reduced from 15.9 hours (serial program) to only 4.3 minutes.

1. Introduction

Over the past few years, building clustering servers for high performance computing is gaining more and more acceptance. Assembling large clusters is easier than ever and the performance is increasing dramatically. According to June 2003's statistical result, there are 149 clusters in the top500 supercomputing machine list [1], while there were 93 clusters among top500 six month ago. Cluster-based solutions for new driving applications need to be developed to take full advantage of the current computing resources. Demanded by highly potential academic research and drug discovery market, bioinformatics application will be an important driving force for the development and deployment of clustering servers and even the design of next generation computer architecture and programming models.

On the other hand, a large amount of DNA, RNA and protein sequences are discovered by different sequencing projects at an accelerating rate in the post genome era. It is very important to find out the structure and function of all the sequences to recover useful information. Sequence

database searching and family classification are common ways to analyze the function of the sequences. The family classification of sequences is of particular interest to drug discovery research. For example, if an unknown sequence is identified to belong to a certain protein family, then its structure and function can be inferred from the information of the family. Furthermore if this sequence is sampled from certain diseases X and belongs to a family F, then X can be treated using the combination of existing drugs for F [2].

Sequence classification is an important computation intensive applications which can exploit the computing resources of supercomputing clusters. A typical way to do sequence classification is to use consensus HMM model (Hidden Markov Model) [3] built from multiple alignment of protein families. HMM is a probabilistic graphical model used very widely in speech recognition and other areas [4]. In recent years, HMM has been an important research topic in bioinformatics area. It is applied systematically to model, align, and analyze entire protein families and secondary structure of sequences. Unlike conventional pairwise comparisons, a consensus model can in principle exploit additional information such as the position and identity of residues that are more or less conserved throughout the family, as well as variable insertion and deletion probabilities [5].

HMM is used to solve three types of questions: 1) How to build a HMM to represent a family? 2) Does a sequence belong to a family? For a given sequence, what is the probability that this sequence has been produced by a HMM model? 3) Assuming a sequence comes from a family represented by a HMM, what can we say about the structure of the sequence? The problem solved in this paper falls into the second category. Usually, for a given unknown sequence, it is necessary to do a database search against a HMM profile database which contains several hundreds of families.

HMMER [6] is an implementation of profile hidden Markov Models (profile HMMs) for sensitive database searches. A wide collection of protein domain models have been generated by using the HMMER package. These mod-

els have largely comprised the Pfam protein family database [7], which is widely used for protein domain detection and function prediction. Hmmpfam, one program in the HMMER 2.2g package, is a widely used tool for searching a single sequence against an HMM database. In real situations, this program may take a few months to finish processing large amounts of sequence data. Thus efficient parallelization of the Hmmpfam is essential to bioinformatics research.

HMMer 2.2g provides a parallel hmmpfam program based on PVM (Parallel Virtual Machine) [8]. However, this PVM version does not have good scalability and can not fully take advantage of the current advanced supercomputing clusters. So a cluster-based solution for hmmpfam is an urgent demand, and it should be highly scalable and robust on hundreds to thousands of computing nodes. To meet such challenges, we exploit the power of a multi-threaded architecture and program execution model, such as EARTH (Efficient Architecture for Running THreads) model [9][10], where parallelism can be efficiently exploited on top of a supercomputing cluster based on off-the-shelf microprocessors.

The major contributions in this paper are as follows: 1) the first EARTH execution model based parallel implementation of bioinformatics sequence classification application; 2) the first largely scalable parallel Hmmpfam implementation targeted to advanced supercomputing clusters; 3) a new efficient master-slave dynamic load balancer is implemented in EARTH Runtime System. This load balance is targeted to parallel applications adopting master-slave model and shows more robust performance than static load balancer.

The rest of paper is organized as follows. In section 2, we review the hmmpfam program and the original parallel scheme implemented on PVM. Our cluster-based multithreaded parallel implementation is described in section 3 and section 4. We present results of our implementation in section 5, and conclusions in section 6.

2. HMMPFAM Algorithm and PVM Implementation

Hmmpfam reads a sequence file and compares each sequence within it, one at a time, against all the HMMs in hmm database, looking for significantly similar matches.

A parallel hmmpfam program based on PVM is included in the HMMer 2.2g. Figure 1 shows the decomposition of the task space of the parallel scheme in the current PVM implementation of hmmpfam. In this scheme, master-slave model is adopted, within one stage, all slave nodes work on the computation for the same sequence. The master node dynamically assigns one profile from the database to a specific slave node, and the slave node is responsible for the

alignment of the sequence to this HMM profile. Upon finishing its job, the slave node reports the results to the master, which will responds by assigning a new job, i.e. a new single profile, to that slave node. When all the computation of this sequence against the whole profile database is completed, the master node sorts and ranks the results it collects, and outputs the top hits. Then the computation on the next sequence begins.

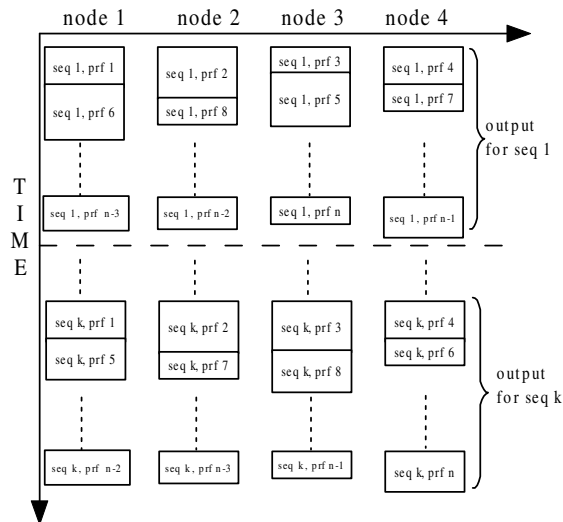


Figure 1. Parallel scheme of PVM version

However, the experimental results shows that this implementation does not achieve good scalability with the number of computing nodes increasing (figure 5, figure 6). The problem is that the granularity of computation is too small relative to the communication overhead. Moreover, the master node becomes a bottleneck with the increase of the number of the computing nodes, since it involves in both communications and local computation such as sorting and ranking. The implicit barrier at the end of the computation of one sequence also wastes the computing resources of the slave nodes.

3. Implementation Strategy On EARTH

The new parallel implementation of the hmmpfam algorithm is based on EARTH multi-threaded architecture, which is developed by CAPSL at the University of Delaware. In this section, before presenting our implementations we briefly describe EARTH, the parallel multi-threaded architecture and execution model. And then we describe our parallelization strategy on EARTH.

3.1. The EARTH Execution Model

EARTH (Efficient Architecture for Running THreads) [9][10] supports a multithreaded program execution model in which a program is viewed as a collection of threads whose execution ordering is determined by data and control dependencies explicitly identified in the program. Threads, in turn are further divided into fibers which are non-preemptive and scheduled according to data-flow like firing rules, i.e., all needed data must be available before it becomes ready for execution. Programs structured using this two-level hierarchy can take advantage of both local synchronization and communication between fibers within the same thread, exploiting data locality. In addition, an effective overlapping of communication and computation is made possible by providing a pool of ready-to-run fibers from which the processor can fetch new work as soon as the current fiber ends and the necessary communication is initiated.

The EARTH architecture executes applications coded in Threaded-C, a multi-threaded extension of ANSI-C programming language, which by incorporating EARTH operations, allow the programmer to indicate the parallelism explicitly. Although designed to deal with multiple threads per node, the EARTH model does not require any support for rapid context switching (since fiber is non-preemptive) and is well-suited to running on off-the-shelf processors. EARTH Runtime System 2.5 (RTS 2.5) is implemented to support the execution of EARTH applications on beowulf clusters that contains SMP nodes.

The EARTH RTS 2.5 [11][12] provides an interface between an explicitly multi-threaded program and a distributed memory hardware platform [13]. It's portable on various platforms: x86 based beowulf cluster, Sun SMP cluster, IBM SP2, etc. It performs fiber scheduling, inter-node communication, inter-fiber synchronization, global memory management, and an important feature – dynamic load balancing, which benefits our second parallel approach.

3.2. Embarrassingly Parallel Computing Strategy

The basic idea of hmmpfam algorithm is to read a single sequence from seqfile each time and to compare it against all the HMMs in the hmmfile looking for significantly similar sequence matches. To efficiently parallelize this application, it is important to determine the granularity of computation. In the original scheme, alignment of one sequence with one profile is considered a single job. By profiling a typical benchmark on a Pentium III 500MHz machine, we find that the average computation time for such a single job is only 0.03 seconds. Thus the granularity of computation

in original parallel scheme is too small relative to the latency and the overheads.

However, if we consider the computation of one sequence against the whole database as a single job, then the job itself is sufficiently big. Normally the number of sequences in a seq file is much larger than the number of computing nodes available in current beowulf clusters. So the number of single jobs is still large to keep all nodes busy, thus to achieve ideal parallelism. Moreover, because all the computation of one single sequence will be performed by one process on one fixed node, there is no need to send back the result to the master node, the sorting and ranking can be executed locally. This means after a job is assigned to one slave node, there is no more communication needed between the master and this slave, there is also no implicit or explicit barriers needed.

Therefore, the hmmpfam algorithm is one that can be immediately divided into completely independent parts that can be executed simultaneously. It is an ideal embarrassingly parallel computation problem, which could be efficiently implemented by classical master-slave parallel model. By using the embarrassingly parallel programming strategy, the new parallel hmmpfam version can take advantage of the current advanced supercomputing clusters to reduce the execution time dramatically with scalable speedup. Following subsections illustrate how the cluster-based solution for parallel hmmpfam is implemented on EARTH.

3.3. A Two Level Master-Slave Parallel Scheme

To exploit the parallelism of hmmpfam algorithm, our multi-threaded parallel scheme divides the work-load at two different levels. In figure 2, each circle represents a THREADED procedure, which is a C function containing local state (function parameters, local variables, and sync slots) and one or more fibers of code. A THREADED procedure is the basic element to be scheduled. Either the programmer or EARTH RTS can determine where (on which node) the procedure get executed.

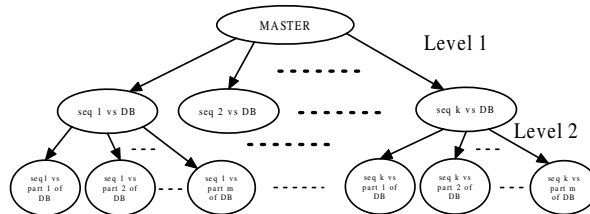


Figure 2. Two level parallel scheme

At the first level, the master process assigns each sequence to one and only one procedure. Each procedure will complete the computation for its sequence against the whole HMM database, then collect and handle the alignment results locally, output the top hits to a file on the local disk. The job at this level is large and independent with other jobs in the same level, so this level exploits the coarse-grain parallelism. The jobs are distributed either manually by programmer or automatically by RTS. Once a job is assigned to a slave procedure, there is no synchronization and barrier required, and no more communication needed between the master and slave processes.

At the second level, in order to achieve further parallelism, the HMM database file is divided to a number of partitions. Each procedure in the second level gets one partition of the database, and performs the corresponding computation, then returns the result to parent procedure. This level exploits the fine-grain parallelism. The jobs are distributed at runtime by the dynamic load balancer of RTS.

4. Cluster-Based Solution

To provide the cluster-based solutions, currently we have successfully implemented the first parallel level in figure 2 by two different approaches: one pre-determines job distribution on all computing nodes by a round-robin algorithm; the other takes advantage of the dynamic load balancing support of EARTH Runtime system, which is more robust and can also simplify the programmer's coding work by making the job distribution completely transparent. Details about those two implementations are described in following sections.

4.1. Static Load Balancing Approach

In the static load balancing implementation, as shown in figure 3, the job distribution is pre-determined. Programmer takes responsibility to explicitly indicate the node, on which a specified job will run. To achieve an even work load on all computing nodes, round-robin algorithm is adopted.

At the initiation stage, master process reads a single sequence from seqfile each time and generates a new job by invoking a threaded procedure on the specified node according to round-robin style. On each computing node, the EARTH RTS maintains a ready queue, which contains the ready-to-run fibers. The RTS will automatically put the first fiber of this procedure to the ready-queue of the node appointed by programmer. As soon as a node gets a new job, it will begin the computation immediately. After the initiation stage, all jobs have been put into the destination computing nodes, so the execution unit on each node can successively fetch new jobs from the ready queue until all are done. That means in the computation stage, all nodes be-

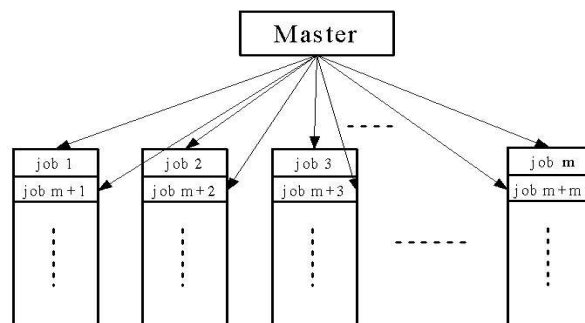


Figure 3. Static load balancing scheme

come independent and execute jobs sequentially without frequently context-switch due to the non-preemptive fiber support of EARTH. If there are very large amount of sequences in the seqfile, which is a very common case, this approach can achieve evenly balanced work load, thus achieve good scalability.

4.2. Robust Dynamic Load Balancing Approach

One of the most important features of EARTH is the dynamic load balancing support. Dynamic load balancer uses system state information to make runtime decision on how to dispatch workloads. The design of dynamic load balancer focuses on two objectives: 1) keeping all the nodes busy; 2) optimizing load balancing by minimizing balancer overheads and maximizing benefits due to load balancing.

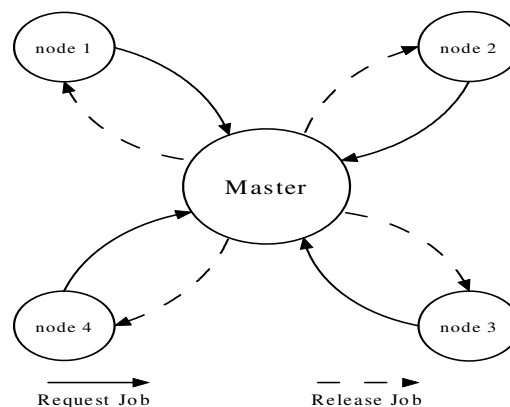


Figure 4. Dynamic load balancing scheme

The target of parallelizing hmmpfam motivates us to design a special load balancer in EARTH RTS 2.5 – master-

slave load balancer, which can be used for all master-slave parallel applications. This load-balancer is working like a server-client model as illustrated in figure 4. Once a slave node finishes a job, it sends a request to the master process. Master responds by sending back a new job to satisfy the slave's work requirement thus to keep it busy. The overhead of load balancing is relatively small since each single job is large enough. With this load balancer, the job-request and job-assignment are determined by EARTH RTS dynamically.

Compared with the pre-determined job assignments strategy, the dynamic load balancing approach is robust. For example, with static load balancing, all jobs have been put into the ready queue of computing nodes at initiation stage, and can not be moved away after that. If one node is busy with computing tasks from other users, the remaining jobs in this node can not be re-assigned to other idle nodes. While with dynamic load balancing strategy, this situation, in which some nodes are busy forever and some others are free, is avoided. Since the hmmpfam tool may run for quite a long time, for instance, several months in real situation, the robust dynamic work distribution strategy is quite practical. Also in order to run the parallel hmmpfam on the advanced supercomputing clusters, which are consisted of hundreds of computing nodes, a robust approach is necessary, since it's not easy to guarantee all nodes working properly without any system problem or the intrusions from other user-level applications during the long running time.

With the dynamic load balancing support of EARTH RTS, the job distribution is completely transparent to the programmer. The programmer only concerns about the generation of new jobs, and does not need to determine where the jobs will be executed. The EARTH RTS takes over the responsibility to distribute jobs at runtime, which quite simplifies and eases programmers' coding work and makes it conceptually straightforward.

5. Experiments and Results

The experiments described in this paper are carried out by using the EARTH Runtime System 2.5 and three different beowulf clusters, two of which are advanced supercomputing clusters in Argonne National Laboratory.

5.1. Computational Platforms

The comparison of PVM hmmpfam version and EARTH version is tested on COMET cluster at CAPSL, University of Delaware. COMET consists of 18 nodes, each containing two 1.4 GHz AMD Athlon processors – total of 36 processors – and 512MB of DDR SDRAM memory. The in-

terconnection network for the nodes is a switched 100Mbps ethernet.

Other experiments are conducted on two large clusters. The Chiba City cluster [14] is a scalability testbed for the High Performance Computing and Computer Science communities at Argonne National Laboratory. The cluster is comprised of 256 computational servers, each with two 500MHz Pentium III processors and 512MB RAM memory. The interconnects for high performance communication are both a fast ethernet and a 64-bit Myrinet.

The JAZZ [15] is a teraflop-class computing cluster, provided by Laboratory Computing Resource Center at Argonne National Laboratory. The JAZZ cluster consists of 350 computing nodes, each with a 2.4 GHz Pentium Xeon processor; and 175 nodes with 2 GB of RAM, 175 nodes with 1 GB of RAM. All nodes are interconnected by fast ethernet and Myrinet 2000.

5.2. The Benchmarks Used in Experiments

For comparison of the PVM version and the EARTH version of parallel hmmpfam, we use a HMM database containing 585 profile families, and a sequence file with 250 sequences. This benchmark is referred as data set-1 in following sections.

For testing both the static and dynamic load balancing version of EARTH hmmpfam, we use a HMM database containing 50 profile families, and a sequence file containing 38192 sequences. This benchmark is referred as data set-2 in following sections.

5.3. Comparison With the PVM Implementation

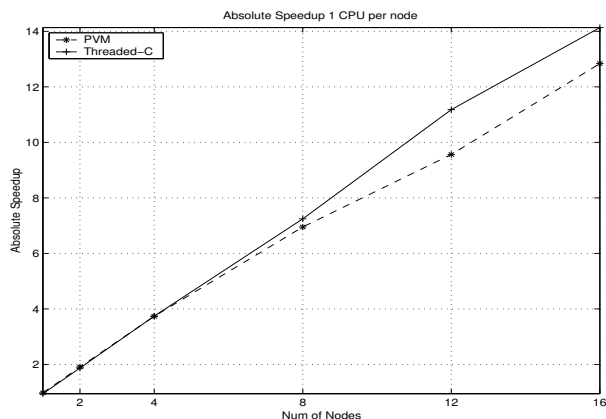


Figure 5. The comparison of PVM version and EARTH version on COMET

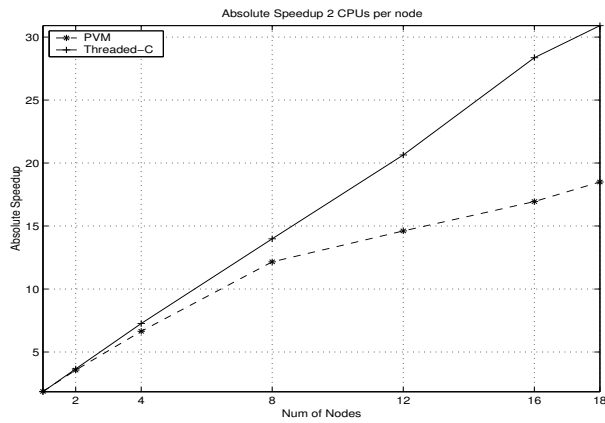


Figure 6. The comparison of PVM version and EARTH version on COMET

First test is conducted to compare the scalability of the PVM version and the EARTH version on COMET Environment using test data set-1. Figure 5 shows the absolute speed up curve when both PVM version and EARTH version are configured to use only 1 CPU per node in COMET, while figure 6 shows the results for dual CPUs per node configuration. From the figures, it is easily seen that our new version has much better scalability, especially in dual-CPU per node configuration. For example, with 16 nodes and 2 CPUs per node configuration, the absolute speedup of the PVM version is 18.50 while the speedup of our version is 30.91, which means 40% reduction of execution time. This is due to the fact that our implementation increases the computation granularity and avoids most communication cost and internal barrier.

5.4. Scalability on Supercomputing Clusters

Second test and third test are conducted to show the performance of our EARTH version hmmpfam on large cluster – Chiba City cluster and Jazz cluster, using test data set-2. The results of both static load balancing and dynamic load balancing schemes are shown in figure 7 to figure 10, where figure 7 and figure 8 show the results for static load balancing on 1 CPU per node and 2 CPUs per node configuration, figure 9 and figure 10 are the results for dynamic load balancing. The two methods do not have much difference in the absolute speedup. This may be due to the fact that subtasks are relatively similar in size, which means static load balancing can also achieve good performance. Both of them show near linear speedup, which means in our new parallel scheme, the serial part only occupies a very small percentage of the total execution. As long as the test data set is big enough, the speedup is expected to keep near linear

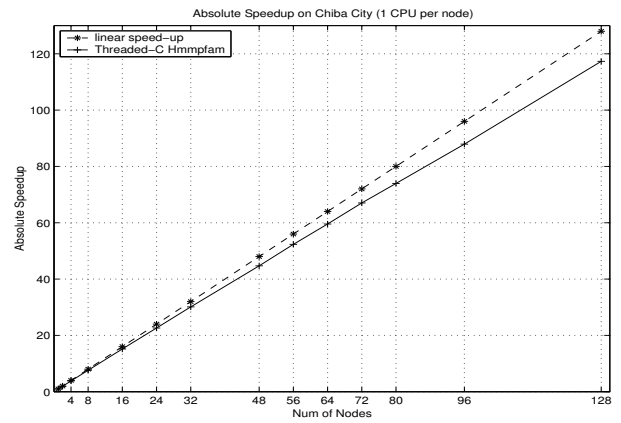


Figure 7. Static load balancing on Chiba City

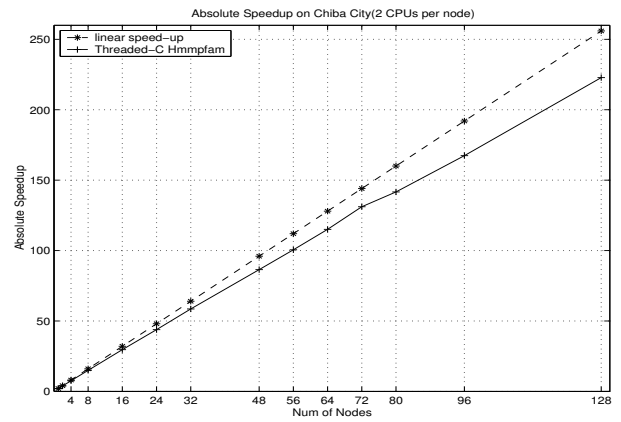


Figure 8. Static load balancing on Chiba City

up to 128 nodes on Chiba City Cluster. The test results on Jazz cluster is shown in figure 11. The speed up curve shows our implementation can still get near linear speed up on 240 nodes, which are with modern INTEL XEON CPUs.

5.5. Robustness Experimental Results

One of the advantages of the dynamic load balancing approach is its robustness. The experiments are conducted to show the program with dynamic load balancing is less affected by the disturbance. The Blastall program is used as the disturbance source since this program is another commonly used computation intensive bioinformatics software.

The execution time for both static and dynamic approach with and without disturbance is measured. Let T denote the execution time without disturbance, and T' denote the execution time with disturbance. Define the *performacne*

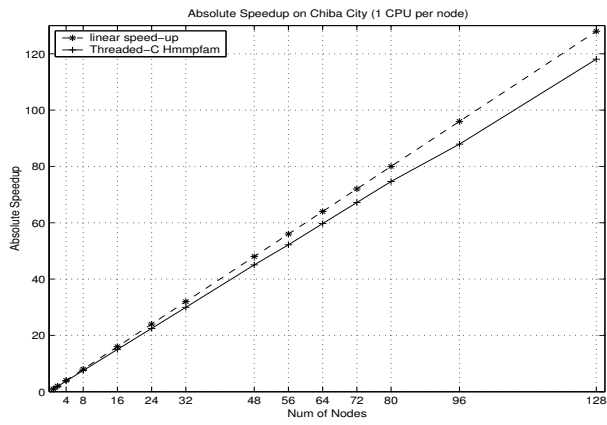


Figure 9. Dynamic load balancing on Chiba City

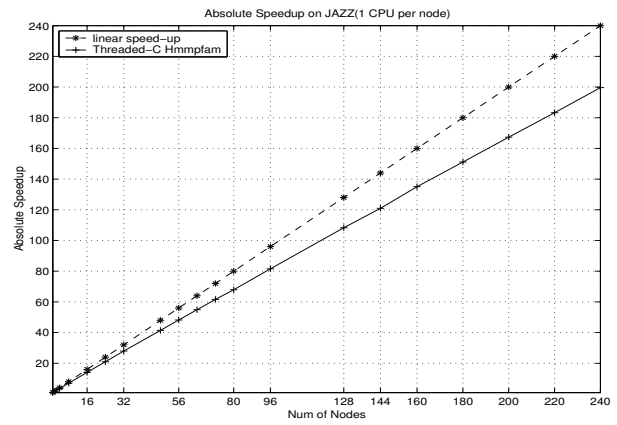


Figure 11. Dynamic load balancing on JAZZ

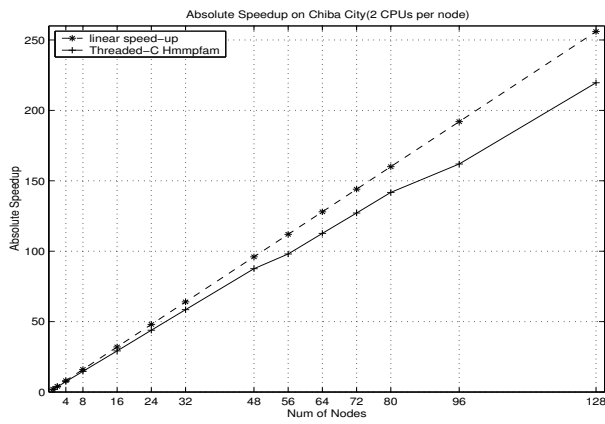


Figure 10. Dynamic load balancing on Chiba City

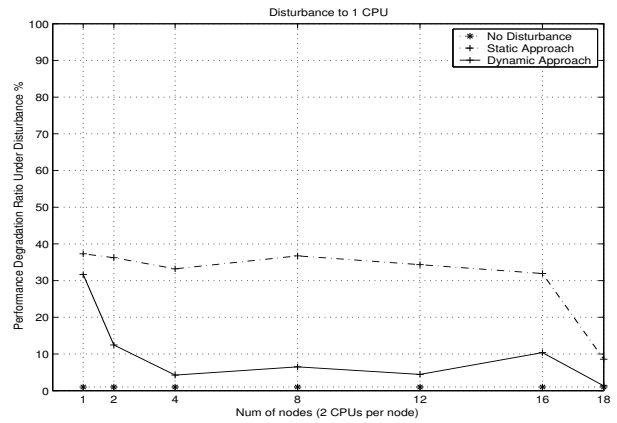


Figure 12. Performance degradation ratio under disturbance to 1 CPU

degradation ratio under disturbance (PDRD) as:

$$PDRD = \left(\frac{T' - T}{T} \right) \times 100\%$$

The PDRD is computed and plotted for both static and dynamic approaches. A smaller PDRD indicates the performance is less influenced by the introduction of disturbance, thus implies the higher implementation robustness.

Figure 12 shows the result when only one blastall program is running on 1 CPU to disturb the execution of hmpfam, and the figure 13 shows result when two CPUs of one node are both disturbed. From the figures, it is apparent that the dynamic load balancing program is less affected by the disturbance and has higher robustness.

6. Conclusions

We implemented a new cluster-based solution of HMM database searching tool on EARTH (Efficient Architecture for Running Threads) and demonstrated significant performance improvement over the original parallel version based on PVM. Our solution provides near linear scalability on supercomputing clusters. On a cluster of 128 dual-CPU nodes, the execution time of a representative testbench is reduced from 15.9 hours to 4.3 minutes. Comparison between the static and dynamic load balancing approach shows the latter is an more robust and practical solution for large time consuming applications running on clusters.

This new implementation could allow researchers to analyze biological sequences at a much higher speed and also make it possible for scientists to analyze problems that were previously considered too large and too time con-

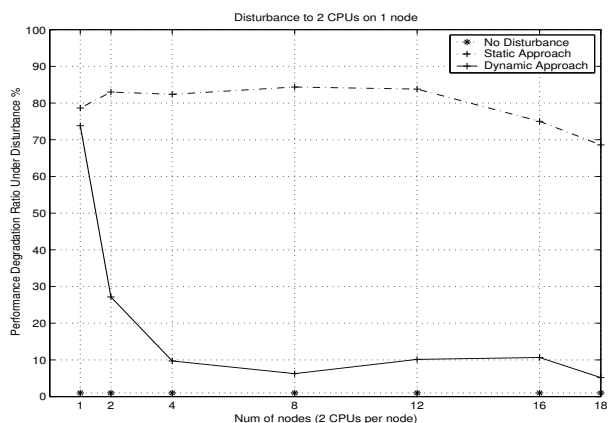


Figure 13. Performance degradation ratio under disturbance to 2 CPUs on 1 node

suming. We presented and evaluated two different parallel schemes which are targeted to advanced supercomputing clusters. One is based on the round-robin algorithm; the other is based on the dynamic load balancing mechanism. The parallelization implementation in this paper motivated the addition of robust dynamic load balancing support into EARTH model, which proves that applications could be the driving force for design of architecture and programming model.

Sequence family classification and HMM database searching is very important to the biological research community. With the help of supercomputing resources and cluster-based solution for applications, researchers can now save research time and get new discoveries more quickly than ever. Many other bioinformatics applications are very time consuming and in essence embarrassingly parallel, which makes them very suitable applications for cluster computing. Porting of hmmpfam to EARTH model provides very promising results, thus further research includes the porting of other bioinformatics applications such as multiple sequence alignment, phylogenetic tree generation to EARTH platform.

7. Acknowledgments

This research is funded in part by NSF, under the NGS grant 0103723, DOE, grant number DE-FC02-01ER25503. We would like to thank the Bioinformatics and Microbial Genomics Team at the Biochemical Sciences and Engineering Division within DuPont CR&D for the useful discussions during the course of this work. We thankfully acknowledge the experiment platform support from the HPC Systems Group and the Laboratory Computing Resource Center at Argonne National Laboratory.

References

- [1] The TOP500 Supercomputer List. [Online]. Available: <http://www.top500.org/>
- [2] C. H. W. Jason T. L. Wang, Qic heng Ma, "Application of neural network to biological data mining: A case study in protein sequence classification," *Proceedings of KDD-2000*, pp. 305–309, 2000.
- [3] S. Eddy, "Profile hidden markov models," *Bioinformatics*, vol. 14, pp. 755–763, 1998.
- [4] M. S.E.Levinson, L.R.Rabiner, "An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition," *Bell Syst.Tech.J.*, vol. 62, pp. 1035–1074, 1983.
- [5] S. B. Pierre Baldi, *Bioinformatics: The machine learning approach*. Cambridge, Massachusetts: The MIT press, 2001.
- [6] HMMER: sequence analysis using profile hidden Markov models. [Online]. Available: <http://hmmer.wustl.edu/>
- [7] C. L. Bateman A, Birney E, "The pfam protein families database," *Nucleic Acids Research*, vol. 30, no. 1, pp. 276–280, 2002.
- [8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.
- [9] K. B. Theobald, "EARTH: An efficient architecture for running threads," Ph.D. dissertation, May 1999.
- [10] H. H. J. Hum, O. Maquelin, K. B. Theobald, X. Tian, X. Tang, and G. R. G. et al., "A design study of the EARTH multiprocessor," *Proceedings of the Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 59–68, 1995.
- [11] C. J. Morrone, "An EARTH runtime system for multiprocessor/multi-node Beowulf clusters," Master's thesis, Univ. of Delaware, Newark, DE, May 2001.
- [12] C. Li, "EARTH-SMP: Multithreading support on an SMP cluster," Master's thesis, Univ. of Delaware, Newark, DE, Apr. 1999.
- [13] H. H. J. Hum, K. B. Theobald, and G. R. Gao, "Building multithreaded architectures with off-the-shelf microprocessors," *In Proceedings of the 8th International Parallel Processing Symposium*, pp. 288–294, 1994.
- [14] The Argonne Scalable Cluster. [Online]. Available: <http://www-unix.mcs.anl.gov/chiba/>
- [15] The Argonne JAZZ Cluster, Laboratory Computing Resource Center (LCRC). [Online]. Available: <http://www.lcrc.anl.gov/jazz/>