Building the Tree of Life on Terascale Systems

Xizhou Feng[1], Kirk W. Cameron[1], Carlos P. Sosa[2], and Brian Smith[2]

[1]Virginia Tech
Dept. of Computer Science
Blacksburg, VA 24061, USA
{fengx, cameron}@cs.vt.edu

[2]IBM Rochester
BlueGene/L Development
Rochester, MN 55901, USA
{cpsosa, smithbr}@us.ibm.com

Abstract

Bayesian phylogenetic inference is an important alternative to maximum likelihood-based phylogenetic method. However, inferring large trees using the Bayesian approach is computationally demanding—requiring huge amounts of memory and months of computational time. With a combination of novel parallel algorithms and latest system technology, terascale phylogenetic tools will provide biologists the computational power necessary to conduct experiments on very large dataset, and thus aid construction of the tree of life.

In this work we evaluate the performance of PBPI, a parallel application that reconstructs phylogenetic trees using MCMC-based Bayesian methods, on two terascale systems, Blue Gene/L at IBM Rochester and System X at Virginia Tech. Our results confirm that for a benchmark dataset with 218 taxa and 10000 characters, PBPI can achieve linear speedup on 1024 or more processors for both systems.

1. Introduction

Phylogeny, a tree or network-like structure representing the evolutionary relationship among a group of species, serves as an important framework to organize, compare, and analyze biological data. Besides its primary role in understanding biological evolution and diversity, it has also been widely used in many other areas including genetics, genomics, drug discovery, plant improvement, and disease control. The importance of phylogeny to science and society can be best demonstrated by the NSF ATOL project [1], whose goal is to provide an overall framework for retrieving, comparing, and predicating huge amounts of biological data by "assembling a tree of life for 1.7 million described species on the earth".

1-4244-0910-1/07/$20.00 ©2007 IEEE

The fundamental task of most phylogenetic inference is to estimate the "correct" phylogenetic trees given one or multiple data sets which encode the clues for the evolutionary path. Among various phylogenetic inference approaches, the Bayesian approach distinguishes itself in several aspects. First, it uses explicit models of evolution and likelihood functions similar to maximum likelihood estimation, another important statistical phylogenetic method. The Bayesian approach has the potential to incorporate complicated models and existing knowledge into the process of phylogenetic inference. Second, it takes a probabilistic view of the estimated trees and ranks these trees with a quantity called posterior probability. Bayesian phylogenetic inference avoids the baffle present in many NP-hard optimality methods that output one "best" tree.

Building large phylogenetic trees using Bayesian approach is computationally demanding. For example, building a phylogenetic tree with hundreds of taxa and thousands of characters may require several gigabytes of memory usage and several months of computing time. To make Bayesian phylogenetic inference more efficient and more practical for large phylogenetic problems, it is necessary to run phylogenetic tools on terascale systems.

The main contributions of this paper are in two folds. First, we provide the excellent scaling results of PBPI, a parallel Bayesian phylogenetic code, on terascale systems. Second, we analyze and compare the performance of PBPI on two different terascale system architectures—the Blue Gene/L system at IBM Rochester and the System X at Virginia Tech. We profile the performance of PBPI on up to 1024 processors on System X and up to 4096 processors on Blue Gene/L. Our results showed that PBPI achieves linear speedup for both fixed-size and increasing workloads on both systems. To the best of our knowledge, this is the first effort to achieve terascale computing capability for Bayesian phylogenetic inference.

This paper is organized as follows. In the next section we review Bayesian phylogenetic inference techniques. Then we describe the parallel strategies used by PBPI in Section 3. Overviews of the Blue Gene/L and System X

are provided in Section 4, following by the experimental results and analysis for both systems. We discuss the related work in Section 5. Finally, we summarize our major conclusions and future work.

2. Bayesian phylogenetic inference

2.1. The problem

Phylogenetic trees reconstruction is one of the grand challenge problems in computational biology. As one of the standard phylogenetic methods, Bayesian phylogenetic inference reconstructs phylogenetic trees from observed data based on the notion of posterior distribution of phylogenetic trees. This method was first developed by several independent research groups (Yang et al at UC Berkeley[2], Mau et al at University of Wisconsin [3], and Li et al at Ohio State [4]) almost simultaneously, and then became widely used after the development of several sequential tools such as BABME [5] and MrBayes [6].

When applied to molecular sequences, the problem of Bayesian phylogenetic inference can be stated as follows:

Given $D = \{d_{ij}\}_{N\times M}$, a matrix of aligned molecular sequences, find a phylogenetic model $\Psi = \Psi(T, \tau, \theta)$ that best interprets (or fits) the data D while providing a confidence measurement for the estimated models.

Here N is the number of taxa (or species); M is the number of site (i.e. characters) in the sequence alignment; d_i denotes the sequence for i^{th} species; and d_{ij} denotes the state of the j^{th} site on the i^{th} sequence. For a DNA sequence, the number of possible states for d_{ij} is 4; for sequences of amino acid or codons, the numbers of possible states are 20 and 60 respectively. A phylogenetic model Ψ describes how the sequences evolve historically. It consists of three components: a tree structure (T) that represents the evolutionary patterns for the species under study, a vector of branch lengths (τ) which maps the divergence time along different lineages, and a model of the molecular evolution (θ) that approximates how the characters at each site evolve over time along the tree.

2.2. The Bayesian approach

The Bayesian approach treats both the observed data D and the phylogenetic model Ψ as random variables and models all random variables using a joint probabilistic distribution, i.e.

$$P(D, \Psi) = P(D \mid \Psi)P(\Psi) . \qquad (1)$$

Once the data is known, Bayesian theory can be used to compute the posterior probability for a specific phylogenetic model Ψ_i using

$$P(\Psi_i \mid D) = \frac{P(D \mid \Psi_i)P(\Psi_i)}{\sum_j \left(P(D \mid \Psi_j)P(\Psi_j)\right)} \qquad (2)$$

Here, $P(D \mid \Psi_i)$ is called the likelihood (the probability of the data under the model Ψ_i); $P(\Psi_i)$ is called the prior probability of the model (the unconditional probability of the model without any knowledge of the observed data); and $P(D) = \sum_j \left(P(D \mid \Psi_j)P(\Psi_j)\right)$ is the unconditional probability of the data.

Since phylogenetic models are hypotheses about how the data will evolve, both the prior and the posterior of a specific phylogenetic model should be interpreted as a confidence interval for the model instead of explained as frequencies. Thus Bayesian theory reflects how the prior belief is updated to posterior belief after having observed new evidence (i.e. data).

In the context of Bayesian phylogenetic inference, the likelihood has the same formulation as the one defined in the maximum likelihood estimation [7]. Borrowing the notation in [8], given a known phylogenetic model $\Psi = \Psi(T, \tau, \theta)$, the probability to observe the u^{th} column of the data matrix D is computed by

$$L(x_u \mid T, \tau, \theta) = \sum_{a^{N+1}\in S} \sum_{a^{N+2}\in S} \cdots \sum_{a^{2N-1}\in S} \left\{ \pi_{a^{2N-1}} \times \prod_{i=N+1}^{2N-1} p\left(a_u^i \mid a_u^{\alpha(i)}, \tau_i, \theta\right) \times \prod_{i=1}^{N} p\left(x_u^i \mid a_u^{\alpha(i)}, \tau_i, \theta\right) \right\} \qquad (3)$$

Here $x_u = \left(d_{ij}\right)\Big|_{j=u}$ is the u^{th} column of D , x_u^i is the state at the u^{th} column on the i^{th} sequence which is mapped to the i^{th} leaf node, $\alpha(i)$ is the index of the parent node of node i and S is the set of possible character states. We can augment the data matrix to include the states at each site for ancient species represented by the internal node. Thus, a_u^i is the state at the u^{th} column on the i^{th} sequence which is mapped to the i^{th} internal node, and $\pi_{a^{2N-1}}$ is the frequence of a^{2N-1} at the root node. The conditional probability $p\left(x_u^i \mid a_u^{\alpha(i)}, \tau_i, \theta\right)$ and $p\left(a_u^i \mid a_u^{\alpha(i)}, \tau_i, \theta\right)$ describe the transition probability from the state $a_u^{\alpha(i)}$ on the parent node to the state a_u^i or x_u^i on the child node after a divergent time τ_i (i.e. the branch length from node i to its parent). The conditional probability $p(a \mid b, \tau_i, \theta)$ is

computed from $e^{Q\tau_i}$ and Q is specified by θ. Since the states for the internal nodes are unknown, Equation (3) summarizes all the possible combinations of the internal nodes to get the site likelihood $L(x_u \mid T, \tau, \theta)$. Assuming the observation at each site is identically independent distributed, the likelihood to observe the entire sequence is:

$$P(D \mid T, \tau, \theta) = \prod_{u=1}^{M} L(x_u \mid T, \tau, \theta) \qquad (5)$$

When a proper prior is specified, the posterior probability of a given phylogenetic model $P(\Psi_i \mid D)$ can be computed by Equations (2)-(4). We also note that this distribution of $P(\Psi \mid D)$ is the basis of Bayesian phylogenetic inference from which a lot of useful information can be obtained. For instance, one important application is to estimate the posterior probability of a specific phylogenetic tree topology by calculating the marginal distribution as

$$P(T_i \mid D) = \iint P(T_i, \tau, \theta \mid D) d\tau d\theta . \qquad (6)$$

2.3. The Markov chain Monte Carlo method

The posterior distribution in Bayesian analysis is usually approximated by a class of methods called Markov chain Monte Carlo (usually abbreviated as MCMC) which simulate random variables from a target distribution, known up to a normalizing constant.

The basic idea of the MCMC methods is first to construct a Markov chain that has the space of the phylogenetic models to be estimated as its state space and the posterior probability distribution of the models as its stationary distribution. Next, we simulate the chain and treat the realization as a representative sample from the posterior probability of the parameters of interest.

In Bayesian phylogenetic inference, the Metropolis-Hasting algorithm [9, 10] and its variants are usually used to construct the chains. Figure 1 shows the basic outline of the Metropolis-Hasting algorithm, in which $\alpha\left(\Psi^{(t)}, \Psi\right)$ is called the acceptance probability that is defined as

$$\alpha(\Psi^{(t)}, \Psi) = \min\left(1, \frac{P(\Psi \mid D)}{P(\Psi^{(t)} \mid D)} \cdot \frac{P(\Psi)}{P(\Psi^{(t)})} \cdot \frac{q(\Psi^{(t)} \mid \Psi)}{q(\Psi \mid \Psi^{(t)})}\right) \qquad (7)$$

Here, $q(\Psi \mid \Psi^{(t)})$ is the proposal probability which can be in any form that satisfies

$$q(\cdot \mid \Psi) > 0 . \qquad (8)$$

In theory, a Markov chain constructed with the Metropolis-Hasting algorithm will converge to a stationary distribution if the chain is irreducible, aperiodic, and possesses a stationary distribution given the chain runs long enough [11]. Thus once we design a Markov chain that satisfies these requirements in Bayesian phylogenetic inference, we can approximate the posterior distribution of phylogenetic models correctly. However, several practical implementation issues exist. For instance, the chain has to overcome the stickiness at local optima and maintain a desired acceptance ratio (the percentage of the proposed states that are accepted) in order to approximate the posterior distribution both efficiently and correctly.

```
1. Initialization Phase
   1.1 t ← 0 ; set starting model Ψ^(0) ← Ψ_0
   1.2 Compute likelihood P(D | Ψ^(0))
2. MCMC sampling phase
   While (t ≤ max imum − generation)
   2.1 Propose a new model Ψ from q(• | Ψ^(t))
   2.2 Compute likelihood P(D | Ψ^(0))
   2.3 Decide/Update next chain state
      2.3.1 Compute the acceptance α(Ψ^(t), Ψ)
      2.3.2 Draw random number u ~ U(0,1)
      2.3.3. If u ≤ α(Ψ^(t), Ψ)
             Then Ψ^(t+1) ← Ψ
             Else Ψ^(t+1) ← Ψ^(t)
   2.4 Move to the next generation: t ← t + 1
```

Figure 1: the Metropolis-Hasting algorithm for Bayesian phylogenetic inference

3. Parallel Bayesian Phylogenetic Inference

3.1. The computational complexity of Bayesian phylogenetic inference

We use "Big O" notation (or asymptotic notation) to describe the computational complexity as the asymptotic behavior of the functions used in Bayesian phylogenetic approach. The likelihood calculation using Equation (3) has $S^{4(N-1)}$ terms, in which S is the number of character states in the evolutionary model. This calculation can be reduced to the order of $O\left(N \cdot S^3\right)$ using the algorithms proposed by Felsenstein [7]—which is equivalent to fill a table of conditional probabilities at the given site on each node in the post-order traversal of the phylogenetic tree.

Assuming a conventional MCMC chain with G generations, the asymptotic behavior of the likelihood evaluation in Bayesian phylogenetic inference is

$$O(G \cdot M \cdot N \cdot S^3) . \qquad (9)$$

Since in the MCMC implementation, the candidate phylogenetic model $\Psi(T,\tau,\theta)$ is only slightly different from the current phylogenetic model $\Psi^{(t)}(T,\tau,\theta)$. On the condition θ is unchanged, only the conditional probability on those nodes appearing in the back tracing path from the affected nodes to the root nodes needs to be recomputed. This property (i.e. likelihood local update) can reduce the number of computations to

$$O(G \cdot M \cdot \log N \cdot S^3). \quad (10)$$

Practical phylogenetic inferences often consider rate variations among sites and use C discrete rate categories to approximate a continuous rate distribution (such as Gamma distribution) [12]. Also some population-based MCMC methods can be used to increase the mixing rate of the MCMC sampling in order to avoid chains being trapped at local optima. Assuming Z chains are used, we can update Equations (9) and (10) to

$$O(G \cdot M \cdot N \cdot S^3 \cdot C \cdot Z) \quad (11)$$

$$O(G \cdot M \cdot \log N \cdot S^3 \cdot C \cdot Z) \quad (12)$$

Assuming α percentage of likelihood calculation belongs to local update and ignoring the cost of other operations such as propose candidate models and decide chain moves, the overall computational complexity of Bayesian phylogenetic inference is approximated by

$$O\left(\left(\alpha \log N + (1-\alpha)N\right) \cdot M \cdot S^3 \cdot C \cdot Z \cdot G\right). \quad (13)$$

Similarly, we can estimate the memory requirement as a multi-dimensional array of doubles with asymptotic order of

$$O(N \cdot M \cdot S \cdot C \cdot Z) \quad (14)$$

Though Equation (13) and (14) indicate Bayesian phylogenetic inference is a polynomial algorithm, as the problem size increases, its computational complexity will become extremely demanding. Further, it is still an open problem to decide the number of generations for a given problem size such that the MCMC chains can reconstruct the tree with sufficient accuracy.

3.2. The challenges to parallel Bayesian MCMC

Based on the discussion in previous section, calculating the likelihood on P processors will have the same order of the computational complexity as that on a single processor. However, it will decrease both time and space complexity by a factor of P if there is enough parallelism to exploit.

Analytically, the likelihood can be parallelized from three dimensions: across the sequence (i.e., M), across the rate category (i.e., C), and across the chains (i.e., Z). Thus, given $P = M \cdot C \cdot Z$ processors, we may reduce the computational complexity in time and space shown in (13) and (14) to

$$O\left(\left(\alpha \log N + (1-\alpha)N\right) \cdot S^3 \cdot G\right) \quad (15)$$

and

$$O(N \cdot S). \quad (16)$$

There are further possibilities to parallel across N and G. Since such parallelisms require dramatic changes of the underlying algorithm, we leave them for future research topic.

The results implied by Equation (15) and (16) appear promising for developing terascale Bayesian phylogenetic inference tools to benefit from a vast number of processors. However, in reality there are several inherent challenges.

As shown in Equation (10) and (12), Bayesian phylogenetic inference may extensively use likelihood local update to significant improve performance. But this technique is not without consequences.

First, local likelihood updates reduce the computation to communication ratio. It is well known that message communications are costly in terms of execution time. The smaller the granularity between two communications, the more difficult it will be to improve parallel performance.

Second, local likelihood update causes load imbalance between computing processors for different chains since the length of the affect path on current proposals differs across chains. Because of load imbalance, the processors with less workload must wait for others to finish, thus reducing efficiency.

Third, the local likelihood updates require memory storage for the conditional probability table of previous phylogenetic models. If we want to swap two models among two processors, we have to exchange this table as a long message.

Fourth, as stated in the Metropolis-Hasting algorithms illustrated in Figure 1, the proposed candidate state may be accepted or rejected. Therefore, we need keep a copy of the conditional probability table for the phylogenetic model at the current generation. Frequent copying of large chunks of memory will decrease the memory access locality and increase average memory access time.

Finally and most importantly, we have to make the parallel implementation highly scalable when the size of the dataset (determined by N and M) increases, or in other words, the parallel implementations have to support fine-grain parallelism such as likelihood evaluation for a given phylogenetic model. However, communicating a phylogenetic tree with hundreds of nodes may require large numbers of multi-kilobyte messages. An efficient, scalable parallel implementation must ensure the incurred communication overhead to transfer messages of such size do not exceed the execution time to compute the likelihood locally.

In the rest of this section, we describe how PBPI will overcome or bypass these challenges.

3.3. The parallel strategies of PBPI

PBPI (Parallel Bayesian Phylogenetic Inference) is a scalable Bayesian phylogenetic inference code which estimates phylogenetic trees from aligned DNA sequences. The initial goal of PBPI is to build a high performance, unified statistical framework for large scale phylogenetic problems [13]. Current PBPI implementation combines multiple MCMC sampling strategies to tackle problems observed in conventional MCMC sampling approaches used in Bayesian phylogenetic inference [14]. PBPI aims to speedup the Bayesian phylogenetic inference from three aspects:

1) Improved MCMC strategies to decrease the maximum number of generations needed in the MCMC sampling, i.e. decreasing G in Equations (11) and (12);

2) Improved sequential performance by reducing the memory footprint and removing unnecessary memory operations to maintain memory access locality; and

3) Exploit multi-level parallelism to make computations scalable to large problems.

In our previous PBPI work, we validated its correctness and performance at scales less up to 64 nodes [15]. We showed the sequential version of PBPI is up to 19 times faster than its best competitor, MrBayes [6], and up to 46 times faster on 64 nodes for a benchmark dataset of 218 taxa and sequence length of 10,000 characters. We also studied the quality of the phylogenetic trees reconstructed with PBPI using simulated study. We found that for the dataset we tested, PBPI can reconstruct the "true" trees at least as accurate as that reconstructed with MrBayes [13].

In this paper, we attempt to analyze the performance of PBPI for terascale systems. Our goal is to determine the inherent limits of PBPI as systems scale beyond 64 nodes on various types of high-end architectures.

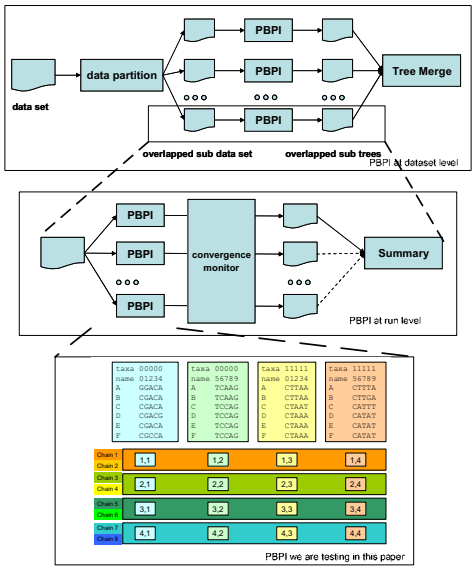

Figure 2: illustration of the parallel schema in PBPI

Figure 2 illustrates the basic parallel schema adopted by PBPI. At the top level, a large dataset is partitioned into multiple overlapped subsets and each individual subset is analyzed by one instance of PBPI. In the middle level, multiple independent runs of PBPI process the same dataset to detect the anomaly or convergence for each run. At the bottom level, PBPI implements fine-grain parallelism at the chain and sequence segment level.

Though it is relatively easier to achieve linear scalability on terascale systems at the top two layers, we will limit our discussion to the effects of fine-grain parallelism. There are two major reasons for this: 1) fine grain parallelism is common to almost all Bayesian phylogenetic inferences; 2) fine grain parallelism is primarily responsible for improving scalability and reducing execution time per MCMC chain generation for a fixed-size problem.

PBPI organizes the processors into a multi-dimensional grid topology and encodes processors in each dimension as a communication group. Similarly, it decomposes the computations during each MCMC chain generation. Thus a one-to-one mapping between each communication group and each level of parallelism is established. For example, Figure 2 illustrates how to map 8 chains onto 16 processors where the number of characters (or patterns) is 20. The 16 processors are arranged as a 4×4 two-dimensional grid where processors at the i^{th} row are grouped as the i^{th} row communication group and processors at the j^{th} row are grouped as the j^{th} row communication group The 8 chains are divided into 4 groups (each group has 2 chains) and the i^{th} chain group is mapped to the i^{th} row communication group. The 20 characters are split into 4 segments and the j^{th} sequence segment is mapped to the j^{th} row communication group.

Besides the IO-related communications to store and display the MCMC states samples, PBPI needs to handle two major kinds of message exchanges: 1) row-wise communications among processors in each row

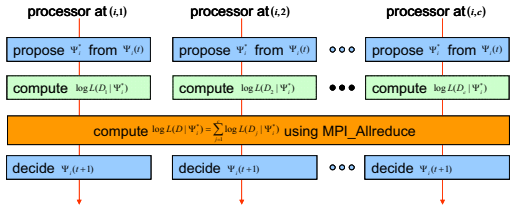

Figure 3: the flow chart of data parallelism during each chain generation. The three different colors represent sequential, parallel, and communication part of the overall workload respectively. In this example, we assume there is only one chain for each row communication group.

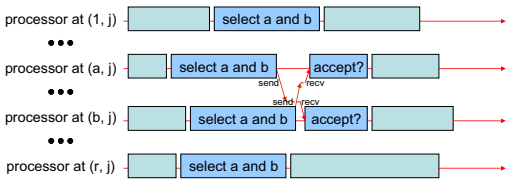

Figure 4: the flow chart of chain level parallelism during each chain generation. In this example, we assume there is only one chain for each row communication group. Due to different number of computations needed by likelihood local update among different chains, the chains among different row communication groups may be not synchronized.

communication group to evaluate, synchronize, and update the states for each MCMC chain, and 2) column-wise communications to swap the states among different chains.

Figure 3 provides a flow chart of data parallelism in PBPI at sequence segment level during each chain generation. Using a row-wise random number generator with the same seeds, the processors at each row communication group duplicate the proposal and decision steps, compute individual partial likelihood for each local dataset (i.e. sequence segment) under the proposed candidate state, and summarize all partial likelihoods to a global likelihood using one collective communication. No further communications and synchronizations are needed to evolve the local Markov chains among processors within the row communication group. Using this schema, PBPI achieves data parallelism among distributed memory systems with minimum communication overhead.

The flow chart of chain level parallelism is shown in Figure 4. Using a grid-wise random number generator with the same seed, the processors within the whole grid select the same pair of chains for swapping. An asynchronous point-to-point communication is performed on the two row communication groups which hold the selected chains. Load imbalance may occur due to different numbers of computations for the likelihood local update. Such imbalance causes processors with fewer computations waiting for data from processors with more computations.

Table 1: the two terascale systems evaluated

System	BG/L	SystemX
Node Technology	IBM BlueGene/L PowerPC 440 System-on-chip	Duel 2.3GHz Apple XServer
# of Processors	8,192	2,200
Peak Performance (in Gflops)	22,938	20,240
LINPACK Perf. (in Gflops)	18,200	12,250

Intuitively, increasing the interval between two chain swapping steps or the number of row groups will reduce the ratio of the average waiting time to the total execution time.

The multi-dimensional domain decomposition leads to a number of factors that influence the achievable performance of PBPI. For instance, by dividing the whole sequence into segments, PBPI is capable of supporting large data set with more complicated evolutionary models while maintaining good cache performance. Furthermore, the average messages exchanged per chain generation is independent to the problem size. In the next section, we use our detailed understanding of the internals of PBPI to analyze performance for PBPI on terascale systems.

4. Experiments and Results

4.1. Comparisons of systems

We compare the performance of PBPI on two tera-scale systems, the IBM BG/L at IBM Rochester and System X at Virginia Tech. Both systems are listed among the top 500 most powerful supercomputers in the world with the performance summarized in Table 1. Since these systems are different in terms of system scale, node architecture, and interconnection technology, we describe the major characteristics below to make the performance comparison more meaningful.

IBM Rochester BG/L System

The Blue Gene/L system we used was one rack of the 4-rack (total 8192 processors) installation at IBM in Rochester. One rack consists of 1024 compute nodes. Each node consists of two PowerPC 440 processor cores, each with a 64-bit floating point unit. Each core runs at 700MHz and each node has 512MB of memory. For benchmarks run in coprocessor mode, the primary core uses the entire 512MB of memory while the secondary core helps with communications operations. For benchmarks run in virtual node mode, each core runs independently, sharing the memory and network resources. Computationally intensive applications may nearly double in performance when run in virtual node mode. See http://www.research.ibm.com/journal/rd/492/moreira.html for further details.

Nodes are connected with a 3D-torus network for point-to-point operations, a global combining tree for reduction operations, and a fast global interrupts network for barrier operations. The torus network has a bandwidth of 10GB/s and a latency of approximately 6 us in the MPI layer. The theoretical peak performance for a rack is 5.6 TFLOPs. Measured LINPACK performance for the 4-rack installation is 18.2 TFLOPs, putting the 4-rack system at number 19 on the Top 500 list.

Table 2: the average execution time in seconds for 10^6 generations on BG/L

Processors	Number of chains				
	4	8	16	32	64
256	1099	2098	4876	10462	22156
512	729	1253	2114	4995	10885
1024	558	869	1173	2164	5124

Table 3: the average execution time in seconds for10^6 generations on System X

Processors	Number of chains				
	4	8	16	32	64
256	550	908	1532	2723	5602
512	479	615	903	1554	2759
1024	440	531	713	961	1572

Virginia Tech System X

System X consists of 1100 Apple XServer G5 cluster nodes. Each cluster node has dual 2.3 GHz PowerPC 970FX processors with 9.2 Gflops peak processor performance. The amount of main memory for each node is 4 GB. Nodes are interconnected with 4× InfiniBand network. The interconnection has a peak inter-node bandwidth of 10 GB/s (bidirectional) and an MPI latency of approximately 7μs.

4.2. Experimental Parameters

We use the 218-taxa backbone tree published by RDP-II projects [16] as the "true" tree and simulate a dataset with 10,000 characters using SEQ-GEN [17] under a JC69 model. We use this dataset as the benchmark dataset and run PBPI on each system then measure the execution time for various configurations using the wall clock time provided by MPI_Wtime( ).

On each system, we investigate both strong scaling and weak scaling. For strong scaling, we fixed the number of chains and increase the number of processors from 256, to 512, and 1024. For weak scaling, we fixed the workload per processor to the size of a row communication group then increase the number of chains from 4, to 8, 16, 32, and 64. In each case, we report results for 1,000,000 chain generations. For each case, we executed 5 runs and report the average execution time over the 5 runs in our results.

Each system configuration is unique. We use dual processors on each node of System X. We run in coprocessor mode on BG/L. We run the code out-of-the-box without further optimization on each system.

We also note direct comparisons of the two systems can be misleading since the architectures are so different. For example, though there are configurations where both machines use the same number of processors, the processors are different both architecturally (superscalar vs. embedded designs) and in operating frequency (700 MHz vs. 2.3 GHz) as well.

4.3. Summary on the results

Table 2 and Table 3 show the raw execution times measured for PBPI on each system for 1,000,000 generations and various chain configurations. For each table, rows correspond to the number of processors used for each machine. Columns reflect the number of chains so that the size of the problem data set increases by a power of two for each column from left to right. As expected, larger problems take longer periods of time. For example on BG/L, 8 chains on 256 processors take about twice as long as 4 chains on 256 processors.

Strong scaling occurs when we fix the workload size and scale the number of processors. Each group of 3 bars in Figure 5 corresponds to a strong scaling study for PBPI on BG/L. In every case, the performance of PBPI improves with an increase in the number of processors. This is also apparent in the raw execution times listed for any single column in Table 3. PBPI was designed to distribute the computational workload evenly across the processors while minimizing the communication overhead. For fixed-size problems, the data distribution improves both local cache and memory performance. This means for the same workload, the local working set is more likely to fit in cache as the number of processors scale. This explains the superlinear performance increases for some of the larger chain configurations on BG/L.

Each group of 3 bars in Figure 6 corresponds to a strong scaling study for PBPI on System X. In every case,

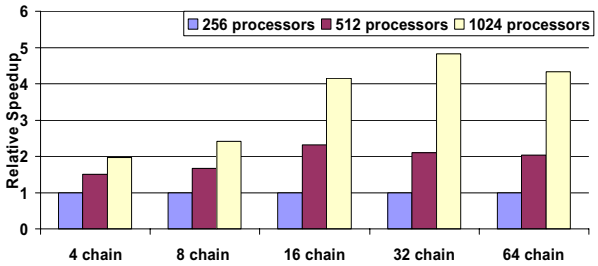

Figure 5: Strong scaling on Blue Gene/L

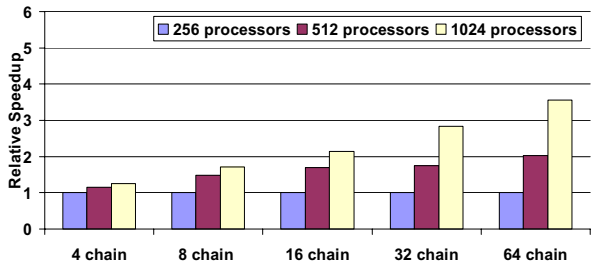

Figure 6: Strong scaling on System X

Table 4: Average GFLOPS on BG/L

	Processors	Number of chains				
		4	8	16	32	64
BG/L	256	45	48	41	38	36
	512	68	80	94	80	73
	1024	89	115	170	184	156

Table 5: Average GFLOPS on System X

	Processors	Number of chains				
		4	8	16	32	64
SystemX	256	91	110	130	146	142
	512	104	162	221	257	289
	1024	113	188	280	415	507

Table 6: The execution time in seconds for 10^6 generations on BG/L (up to 4096 processors)

Processors	Number of chains			
	32 -CO	32-VN	64-CO	64-VN
1024	2491	2457	5023	5239
2048	1379	1314	2662	2498
4096		765		1359

Table 7: Average GFLOPS on BG/L (up to 4096 processors)

Processors	Number of chains			
	32 -CO	32-VN	64-CO	64-VN
1024	160	162	159	152
2048	289	303	300	319
4096		521		587

the performance of PBPI improves with an increase in the number of processors. This is also apparent in the raw execution times listed for any single column in Table 3. Speedups on System X are also encouraging for reasons similar to those discussed for BG/L. However, the speedup effects observed on BG/L are dampened somewhat on System X due to differences in interconnect performance and communication/computation ratio between the two machines. The System X interconnect has larger observable latencies that contribute to increased communication overhead. This overhead lessens the effects of the cache on execution time and speedup.

Weak scaling occurs when we vary the workload size while we scale the number of processors. In our experiments, we fix the workload per processor and then vary the processor configuration. For example, using P as the workload per processor, for 256 processors we scale the workload to 256P; for 1024 processors we scale the workload to 1024P; etc. An optimal result for weak scaling is to maintain a constant execution time as the workload scales. Figures 7 and 8 show PBPI maintains nearly constant execution time for our weak scaling studies. Again this is primarily due to our algorithm design and the effects of workload distribution and small memory working set for large system configurations.

Speedup results can be misleading. Only total execution time for the various workload configurations can determine the best measured performance. As Table 2 and Table 3 show System X results in faster execution time in all cases primarily due to the speed of its G5 processor (2.4 GHz) verse the BG/L IBM processor (700MHz) since PBPI is a computationally intensive code with a relatively small memory working set for large node configurations.

Following common practices, we calculate the average number of floating point operations per second (FLOPS) on both systems by dividing the average execution time in seconds by the total number of floating point operations completed for each run. Though we use GFLOPS to report achieved performance, this metric is not ideally suited to reflect PBPI performance. PBPI performs a significant number of integer operations including generating random numbers and proposing candidate trees. Thus poor or exceptional integer performance on a system may influence results not reflected in GFLOPS.

Tables 4 and 5 provide the sustained performance in GFLOPS for PBPI on various system and workload configurations. For 1024 processors, BG/L sustains 156 GFLOPS while System X sustains 507 GFLOPS (or 0.5 TFLOPS). These results explain some of the observations made in execution time and speedup trends. The higher GFLOP throughput on System X corresponds to its reduced execution time. For both systems, the GFLOP throughput increases with number of nodes for a fixed problem size. For larger system configurations on both

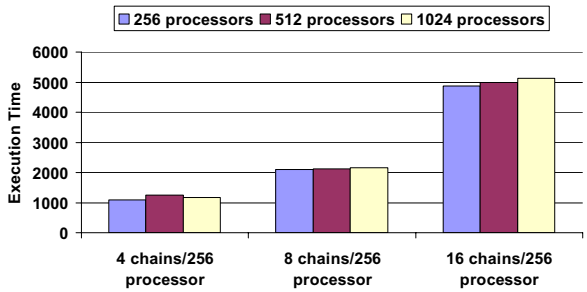

Figure 7: Weak scaling on Blue Gene/L

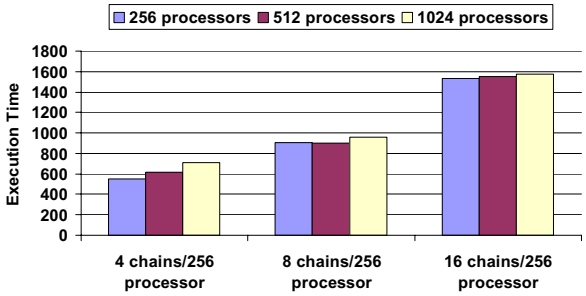

Figure 8: Weak scaling on System X

systems, the GFLOP throughput increases with increases in the problem size. This supports the conjecture that the memory working set is more likely to fit in cache for larger system configurations. This effectively reduces the average memory latency per FLOP, thus increasing the measured GFLOPS.

As we mentioned, direct machine comparisons across architectures are not particularly fair. For example, in our experiments thus far, System X outperformed BG/L for raw execution time. Yet these experiments assume processors are of similar performance. Of course, BG/L was designed to employ larger processor configurations. To this end, we performed an additional study to see how well PBPI scales to extreme processor counts – limited only by our ability to secure time on large configurations of BG/L.

Table 6 and Table 7 show data for various additional configurations of BG/L up to 4096 processors. This time, we ran in both coprocessor (CO) and virtual node mode (VN) for 32 and 64 chains. PBPI achieves roughly same performance on both modes for the same run configuration. As VN mode doubles the number of processors, VN mode provides PBPI about 1.7~2.0 times performance gain than CO mode on the same number of nodes. Figure 9 shows the results for strong scaling up to 4096 processors (256, 512, and 1024 are in coprocessor mode; 2048 and 4096 are in virtual node mode). This figure indicates that the scaling effects observed continue beyond 1024 processors. The 32 chain workload appears to be losing some scalability since linear speedup (not superlinear as before) is observed from 2048 to 4096 processors. For the larger data set of 64 chains, super linear speedup continues through 4096 processors. We also note that for both 32 and 64 chains, the 4096 processor configuration is able to sustain more than 500 GFLOPS (0.5 TFLOPS).

In summary, our results indicate the PBPI scales superlinearly for larger processor configurations and workloads on two distinct terascale architectures. This is primarily due to an efficient algorithm that distributes workloads evenly and results in smaller working sets that fit in cache on large system configurations. Additionally, the performance of PBPI is particularly sensitive to processor frequency. We also observed superlinear speedup continues (albeit for large workloads only) on extreme processor counts.

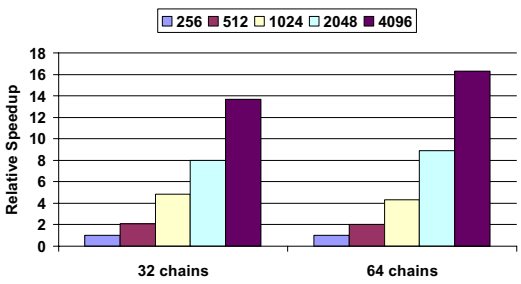

Figure 9: Strong scaling on Blue Gene/L up to 4096 processors

5. Related work

Several scalable parallel programs for maximum likelihood-based phylogenetic methods, have been developed recently, among them are parallel fastDNAml [18, 19], PAxML [20], parallel genetic algorithm for ML [21], and parallel TREE-PUZZLE [22]. Bader et al discussed GRAPPA, a highly scalable implementation for breakpoint phylogenetic analysis using gene order data [23]. Most of these codes are targeted at searching the tree space and there is no dependency in the time dimension. Thus by carefully dividing the search space into multiple subspaces, the parallel code can achieve coarse granularity parallelisms and use the master-slave parallel schema in the implementation. Comparing with this code, Bayesian phylogenetic inference uses different criterion to define the "optimal" trees for given datasets and samples trees from the tree space according to the probabilities distribution of the trees. There is a strong time dependency between two adjacent samples. Therefore to achieve strong scaling we have to parallel at much finer granularity.

MrBayes also provided a parallel code for Bayesian phylogenetic inference (see Altekar et al [24]). Its major issue lies in the fact that the code can only parallel at chain level. The implication is that given a fixed problem and fixed number of chains (for example 4 chains), it can not run more than 4 nodes. Another issue is the code may fail when the memory requirement for one chain is larger than that can be supported by a single node. PBPI overcomes this limitation by combining sequence level parallelism and chain level parallelism. Another key difference between PBPI and MrBayes is that PBPI implemented several improved MCMC strategies to overcome a danger of Metropolis-coupled MCMC chains may be trapped to a set of local optima and fail to explore the space properly. Further, PBPI achieved significant improvements in sequential code due to reduced memory footprint and a reduced amount of memory copy operations.

6. Summary and future work

Exploiting the power of terascale systems to tackle the computational challenges in Bayesian phylogenetic tree reconstruction is necessary for various reasons. For instance, it allows biologists to analyze large data sets to gain further insight into evolution and biological diversity with improved accuracy. We also described the asymptotic behavior of PBPI and the analytical impact of various design decisions on performance. We demonstrated the effects of these optimizations on two representative terascale systems, IBM Blue Gene/L and Virginia Tech System X. The experimental results demonstrate PBPI can

achieve excellent strong and weak scaling up to thousands of processors on both systems we tested. We also noted some interesting similarities and differences specific to the two systems.

As future work, we hope to continually improve the PBPI framework. Future versions will include additionally supported models of evolution and more advanced MCMC algorithms. We also plan to optimize the code for better cache efficiency and better floating point performance. Furthermore, since the performance results for PBPI are exceptional, we are working with biologists to identify challenging data sets for analyses where identification of phylogenetic trees can lead to important biological insights as to the origin of species.

References

[1] U.S. National Science Foundation, "Assembling the Tree of Life (ATOL): To construct a phylogeny for the 1.7 million described species of life," National Science Foundation, Program Solicitation, NSF 04-526, 2004.

[2] Z. H. Yang and B. Rannala, "Bayesian phylogenetic inference using DNA sequences: A Markov Chain Monte Carlo method," *Molecular Biology and Evolution,* vol. 14, pp. 717-724, 1997.

[3] B. Mau, M. A. Newton, and B. Larget, "Bayesian phylogenetic inference via Markov chain Monte Carlo methods," *Biometrics,* vol. 55, pp. 1-12, 1999.

[4] S. Y. Li, D. K. Pearl, and H. Doss, "Phylogenetic tree construction using Markov chain Monte Carlo," *Journal of the American Statistical Association,* vol. 95, pp. 493-508, 2000.

[5] D. L. Simon and B. Larget, "Bayesian analysis in molecular biology and evolution (BAMBE)," Department of Mathematics and Computer Science, Dequesne University, 1998.

[6] J. P. Huelsenbeck and F. Ronquist, "MRBAYES: Bayesian inference of phylogenetic trees," *Bioinformatics,* vol. 17, pp. 754-755, 2001.

[7] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach," *Journal of Molecular Evolution,* vol. 17, pp. 368-76, 1981.

[8] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*: Cambridge University Press, 1998.

[9] N. Metropolis, A. N. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of state calculations by fast computing machine," *J. Chem. Phys.,* vol. 21, pp. 1087-1091, 1953.

[10] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their application," *Biometrika,* vol. 57, pp. 97-109, 1970.

[11] L. Tierney, "Markov-Chains for Exploring Posterior Distributions," *Annals of Statistics,* vol. 22, pp. 1701-1728, 1994.

[12] Z. Yang and S. Kumar, "Approximate methods for estimating the pattern of nucleotide substitution and the variation of substitution rates among sites," *Molecular Biology and Evolution,* vol. 13, pp. 650-659, 1996.

[13] X. Feng, "High Performance, Bayesian-based Phylogenetic Inference Framework," Ph.D. Dissertation, Department of Computer Science and Engineering, University of South Carolina, 2006.

[14] X. Feng, D. A. Buell, J. R. Rose, and P. J. Waddell, "Parallel algorithms for Bayesian phylogenetic inference," *Journal of Parallel and Distributed Computing,* vol. 63, pp. 707-718, 2003.

[15] X. Feng, K. W. Cameron, and D. A. Buell, "PBPI: a High Performance Implementation of Bayesian Phylogenetic Inference," in *The ACM/IEEE SC2006 Conference on High Performance, Networking and Computing*, Tampa, FL, 2006.

[16] J. R. Cole, B. Chai, R. J. Farris, Q. Wang, S. A. Kulam, D. M. McGarrell, G. M. Garrity, and J. M. Tiedje, "The Ribosomal Database Project (RDP-II): sequences and tools for high-throughput rRNA analysis," *Nucl. Acids Res.,* vol. 33, pp. D294-296, 2005.

[17] A. Rambaut and N. C. Grassly, "Seq-Gen: An application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic frees," *Computer Applications in the Biosciences,* vol. 13, pp. 235-238, 1997.

[18] C. A. Stewart, D. Hart, D. K. Berry, G. J. Olsen, E. Wernert, and W. Fischer, "Parallel implementation and performance of fastDNAml - a program for maximum likelihood phylogenetic inference," in *The ACM/IEEE SC2001*, Denver, 2001.

[19] G. J. Olsen, H. Matsuda, R. Hagstrom, and R. Overbeek, "fastDNAmL: a tool for construction of phylogenetic trees of DNA sequences using maximum likelihood," *Comput. Appl. Biosci.,* vol. 10, pp. 41-48, 1994.

[20] A. Stamatakis, T. Ludwig, and H. Meier, "RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees," *Bioinformatics,* vol. 21, pp. 456-463, 2005.

[21] M. J. Brauer, M. T. Holder, L. A. Dries, D. J. Zwickl, P. O. Lewis, and D. M. Hillis, "Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference," *Molecular Biology and Evolution,* vol. 19, pp. 1717-1726, 2002.

[22] H. A. Schmidt, K. Strimmer, M. Vingron, and A. von Haeseler, "TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing," *Bioinformatics,* vol. 18, pp. 502-504, 2002.

[23] B. M. E. Moret, D. A. Bader, and T. Warnow, "High-performance algorithm engineering for computational phylogenetics," *Journal of Supercomputing,* vol. 22, pp. 99-110, 2002.

[24] G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist, "Parallel metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference," *Bioinformatics,* vol. 20, pp. 407-415, 2004.