BIOINFORMATICS

Vol. 20 no. 3 2004, pages 407–415
DOI: 10.1093/bioinformatics/btg427

Parallel Metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference

Gautam Altekar[1,], Sandhya Dwarkadas[1], John P. Huelsenbeck[2] and Fredrik Ronquist[3]*

[1]Department of Computer Science, University of Rochester, [2]Section of Ecology, Behavior and Evolution, Division of Biological Sciences, University of California, San Diego and [3]Department of Systematic Zoology, Evolutionary Biology Centre, Uppsala University

Received on August 29, 2002; revised on April 3, 2003; accepted on April 17, 2003
Advance Access Publication January 22, 2004

ABSTRACT

Motivation: Bayesian estimation of phylogeny is based on the posterior probability distribution of trees. Currently, the only numerical method that can effectively approximate posterior probabilities of trees is Markov chain Monte Carlo (MCMC). Standard implementations of MCMC can be prone to entrapment in local optima. Metropolis coupled MCMC [$(MC)^3$], a variant of MCMC, allows multiple peaks in the landscape of trees to be more readily explored, but at the cost of increased execution time.

Results: This paper presents a parallel algorithm for $(MC)^3$. The proposed parallel algorithm retains the ability to explore multiple peaks in the posterior distribution of trees while maintaining a fast execution time. The algorithm has been implemented using two popular parallel programming models: message passing and shared memory. Performance results indicate nearly linear speed improvement in both programming models for small and large data sets.

Availability: MrBayes v3.0 is available at http://morphbank.ebc.uu.se/mrbayes/

Contact: galtekar@cs.rochester.edu

INTRODUCTION

Bayesian inference is a recently described method for estimating phylogenetic trees (Li, 1996; Mau, 1996; Mau and Newton, 1997; Mau *et al*., 1999; Larget and Simon, 1999; Newton *et al*., 1999; Rannala and Yang, 1996; Yang and Rannala, 1997) that is based on the posterior probability distribution of the trees. The posterior probability of a tree is the probability of the tree relating the species conditional on the phylogenetic observations (such as an alignment of DNA sequences) and is proportional to the likelihood times the prior probability of that tree. In a Bayesian phylogenetic analysis, all estimates of phylogeny are based on the posterior probability distribution of trees. For example, the tree with the maximum posterior probability (MAP) might be chosen as a point estimate of phylogeny (Rannala and Yang, 1996), a 95% credible set of trees might be constructed by ordering trees from highest to lowest posterior probability and, starting with the MAP tree, including trees in a set until the cumulative probability is 0.95 (Felsenstein, 1968), or the posterior probabilities of individual clades (subtrees) might be shown on a majority rule consensus tree or on the MAP tree (Larget and Simon, 1999).

A Bayesian analysis of phylogenetic trees requires the evaluation of high-dimensional summations and integrals. Minimally, the posterior probability of a phylogenetic tree involves a summation over all possible trees. The number of trees for even moderately-sized problems is very large; for example there are about two million unrooted trees possible for 10 species and over a mole (6.02×10^{23}) of trees possible for 22 species. Moreover, for each tree that is considered, the likelihood involves a multidimensional integral over all possible combinations of branch lengths and substitution model parameters (e.g. parameters that allow different rates among the different character states, different stationary character-state frequencies, or rate variation across sites). By necessity, then, posterior probabilities of trees must be approximated. Markov chain Monte carlo (MCMC; Metropolis *et al*., 1953; Hastings, 1970) has been successfully used to approximate the posterior probability distribution of trees (Yang and Rannala, 1997; Larget and Simon, 1999; Huelsenbeck and Ronquist, 2001). MCMC uses stochastic simulation to obtain a sample from the posterior distribution of trees; inferences are then based on the MCMC sample.

The posterior probability distribution of trees can contain multiple peaks. The peaks represent trees of high probability separated from other peaks by valleys of trees with low probability. This is a phenomenon that has been observed for other optimality criteria, such as maximum parsimony (Maddison, 1991) and maximum likelihood (see Salter and Pearl, 2001). Like heuristic searches commonly used to find optimal trees,

*To whom correspondence should be addressed.

MCMC can be prone to entrapment in local optima; a Markov chain currently exploring a peak of high probability may experience difficulty crossing valleys to explore other peaks. A properly constructed Markov chain will eventually cross even very deep valleys in the posterior probability distribution of trees (Tierney, 1994). However, it may take a prohibitive amount of time to adequately explore a rugged landscape of trees. As a result, a large number of trees of high posterior probability may go unexplored in a standard MCMC analysis.

A number of methods can be used to improve the mixing (ability to explore the posterior distribution) of MCMC. Metropolis coupled MCMC [or $(MC)^3$] appears to be an effective method for improving the mixing of MCMC (Geyer, 1991; Gilks and Roberts, 1996) and has been shown to improve convergence by Huelsenbeck *et al*. (2001). $(MC)^3$ involves running multiple chains, some of which are 'heated', and attempting swaps of the states of chains. A heated chain sees the landscape of trees as flattened relative to the 'cold' (or unheated) chain. Hence, heated chains can more readily cross valleys in the landscape of trees. Occasionally, the cold chain will successfully exchange states with a heated chain that may be exploring another peak in the landscape of trees. This allows the cold chain to effectively jump a deep valley in a single step. Despite the improved mixing of $(MC)^3$, the use of multiple chains incurs a significant performance cost. Specifically, MCMC requires the execution of only one chain, but $(MC)^3$ requires the execution of several chains. Since each iteration within a chain involves the calculation of a potentially expensive likelihood function, running several chains increases execution time considerably.

The probability of a Markov chain moving to any particular state in the next generation depends only upon the current state of the chain. This Markov property makes significant parallelization of MCMC algorithms difficult. The limited parallelism of a single chain persists even when running multiple independent chains, but the overhead of running the heated chains can theoretically be nearly eliminated by running them in parallel with the cold chain.

In this paper, we present a parallel $(MC)^3$ [or $p(MC)^3$] algorithm that achieves near optimal speedups with both small and large data sets. The $p(MC)^3$ algorithm constitutes the core of a parallel version of the program MrBayes (v3.0 Huelsenbeck and Ronquist, 2001; Ronquist and Huelsenbeck, 2003), a phylogenetic application that infers the evolutionary history of a group of species using Bayesian inference. We implement the $p(MC)^3$ algorithm in two different programming models: message passing and shared memory. Although the underlying intuition of the algorithm is similar in both models, differences in parallel programming constructs make the two implementations somewhat different. We provide the necessary details to implement each version of the algorithm.

Both implementations of our algorithm are evaluated on a cluster of shared memory (symmetric) multiprocessors (SMPs). The message passing implementation makes use of the message passing interface (MPI) (MPIF, 1994) for sending messages to and from processes. The shared memory implementation uses Cashmere (Stets *et al*., 1997), a software distributed shared memory (SDSM) system developed at the University of Rochester, which provides the illusion of shared memory in software across machines in the cluster. SDSM is able to take advantage of hardware support (thereby providing better performance) for sharing within each SMP, while allowing seamless expansion (without much loss in performance) across SMPs. Message passing has the advantage of allowing applications to minimize overhead when communicating across SMPs by sending only the necessary data. In the message passing implementation, we experiment with two different network technologies: a standard Ethernet network and the high speed Memory Channel network.

BACKGROUND

Bayesian estimation of phylogeny

The object of phylogenetic analysis is to estimate the phylogenetic history for a group of species. Phylogenies are bifurcating trees, and are denoted $\tau_1, \tau_2, \tau_3, \ldots, \tau_{B(s)}$, where $B(s)$ is the number of possible trees for s species [$B(s) = (2s-3)!/[2^{s-2}(s-2)!]$ for rooted trees and $B(s) = (2s-5)!/[2^{s-3}(s-3)!]$ for unrooted trees]. Each branch of the tree has a length, which indicates the expected number of state changes per character along that branch. The set of branch lengths for the i-th tree is denoted $\boldsymbol{v}_i$. Character transformations along the tree are described by a continuous-time Markov process, the parameters of which are contained in a vector $\boldsymbol{\theta}$. Samples taken from the Markov chain are valid, albeit dependent, samples from the distribution of interest (Tierney, 1994).

In a Bayesian analysis, estimates are based upon the posterior probability distribution of a parameter. For the phylogeny problem, estimates are based on the joint posterior probability distribution of $\boldsymbol{\psi} = (\tau, \boldsymbol{v}, \boldsymbol{\theta})$, $f(\boldsymbol{\psi}|\boldsymbol{X})$, which is calculated using Bayes's formula as:

$$f(\boldsymbol{\psi}|\boldsymbol{X}) = \frac{f(\boldsymbol{X}|\boldsymbol{\psi})f(\boldsymbol{\psi})}{f(\boldsymbol{X})}.$$

The Metropolis–Hastings algorithm (Metropolis *et al*., 1953; Hastings, 1970; Green, 1995), a variant of MCMC, works as follows:

(1) Let $\boldsymbol{\psi}$ denote the current state of the Markov chain. If this is the first iteration, $\boldsymbol{\psi}$ is initialized to some value, perhaps a randomly chosen one.

(2) Randomly propose a new value for $\boldsymbol{\psi}$, denoted $\boldsymbol{\psi}'$. The probability of proposing the new state is $q(\boldsymbol{\psi}')$ whereas the probability of proposing the old state conditional on starting at the new state (a move that is not actually made) is $q(\boldsymbol{\psi})$.

(3) The probability, R, of accepting the new state is

$$R = \min\left[1, \frac{f(\boldsymbol{X}|\boldsymbol{\psi}')}{f(\boldsymbol{X}|\boldsymbol{\psi})} \times \frac{f(\boldsymbol{\psi}')}{f(\boldsymbol{\psi})} \times \frac{q(\boldsymbol{\psi})}{q(\boldsymbol{\psi}')}\right].$$

(4) Generate a random variable, U, that is uniformly distributed on the interval $(0, 1)$. If U is less than R, then accept the proposed state and let $\boldsymbol{\psi} = \boldsymbol{\psi}'$. Otherwise, continue with the current state.

(5) Go back to step 2.

This process is repeated for a sufficiently large number of iterations. As long as the chain is properly constructed [i.e. the proposal mechanism(s) from step 2 result in an irreducible and aperiodic Markov chain and there are no programming errors], the long-run frequencies of states visited by the Markov chain will approximate the posterior distribution.

Metropolis coupled MCMC

Metropolis coupled MCMC is a variant of MCMC in which n chains are run (Geyer, 1991). Some of these chains are 'heated' by raising the posterior probability to a power β. For example, if $f(\boldsymbol{\psi}|\boldsymbol{X})$ is the posterior probability density distribution of the phylogenetic parameters, then a heated version of the posterior distribution is $f(\boldsymbol{\psi}|\boldsymbol{X})^{\beta}$. Here, β $(0 < \beta < 1)$ is the heat value of the chain.

Heating a Markov chain increases the acceptance probability of new states. Consider a Markov chain with state $\boldsymbol{\psi}$ and a proposed state $\boldsymbol{\psi}'$. The probability of accepting $\boldsymbol{\psi}'$ for an unheated or 'cold' chain is

$$R = \min\left[1, \frac{f(\boldsymbol{X}|\boldsymbol{\psi}')}{f(\boldsymbol{X}|\boldsymbol{\psi})} \times \frac{f(\boldsymbol{\psi}')}{f(\boldsymbol{\psi})} \times \frac{q(\boldsymbol{\psi})}{q(\boldsymbol{\psi}')}\right].$$

The probability of accepting $\boldsymbol{\psi}'$ for a heated chain, on the other hand, is

$$R = \min\left[1, \left(\frac{f(\boldsymbol{X}|\boldsymbol{\psi}')}{f(\boldsymbol{X}|\boldsymbol{\psi})} \times \frac{f(\boldsymbol{\psi}')}{f(\boldsymbol{\psi})}\right)^{\beta} \times \frac{q(\boldsymbol{\psi})}{q(\boldsymbol{\psi}')}\right].$$

If $f(\boldsymbol{\psi}|\boldsymbol{X}) > f(\boldsymbol{\psi}'|\boldsymbol{X})$, raising each to the power β increases R. Consequently, a heated chain tends to accept more states than a cold chain, allowing a heated chain to more readily cross valleys in the landscape of trees.

$(MC)^3$ involves running n Markov chains where each chain is labeled $i \in (1, 2, \ldots, n)$. We use incremental heating, where the heat for the i-th chain is $\beta_i = 1/[1 + \Delta T \times (i - 1)]$, and $\Delta T > 1$ is a temperature increment parameter (Geyer, 1991). The ΔT parameter is chosen such that swaps are accepted between 20 and 60% of the time, thereby providing a sufficient amount of mixing and justifying the use of $(MC)^3$. Notice that the heating parameter for the first chain is 1 (i.e. $\beta_1 = 1$). For this chain, acceptance probabilities are unaltered, thereby making it the only cold chain.

Like MCMC, $(MC)^3$ is an iterative algorithm, which works as follows:

(1) Let $\boldsymbol{\psi}_i$ denote the current state of Markov chain i. If this is the first iteration, $\boldsymbol{\psi}_i$ is initialized to some value, perhaps a randomly chosen one. This is done for all n chains.

(2) For all chains, $i \in (1, 2, \ldots, n)$

(a) Randomly propose a new value for $\boldsymbol{\psi}_i$, called $\boldsymbol{\psi}_i'$.

(b) Accept or reject $\boldsymbol{\psi}_i'$ with probability R_i,

$$R_i = \min\left[1, \left(\frac{f(\boldsymbol{X}|\boldsymbol{\psi}_i')}{f(\boldsymbol{X}|\boldsymbol{\psi}_i)} \times \frac{f(\boldsymbol{\psi}_i')}{f(\boldsymbol{\psi}_i)}\right)^{\beta_i} \times \frac{q(\boldsymbol{\psi}_i)}{q(\boldsymbol{\psi}_i')}\right].$$

(c) Generate a random variable, U, that is uniformly distributed on the interval $(0, 1)$. If U is less than R_i, then accept the proposed state $\boldsymbol{\psi}_i'$. Otherwise, continue with the current state $\boldsymbol{\psi}_i$.

(3) After all chains have advanced a given number of iterations (say one cycle), two randomly chosen chains j and k are selected to exchange states. The swap of states is accepted with probability

$$R = \min\left[1, \frac{f(\boldsymbol{\psi}_k|\boldsymbol{X})^{\beta_j} f(\boldsymbol{\psi}_j|\boldsymbol{X})^{\beta_k}}{f(\boldsymbol{\psi}_j|\boldsymbol{X})^{\beta_j} f(\boldsymbol{\psi}_k|\boldsymbol{X})^{\beta_k}}\right].$$

(4) A uniformly distributed random number on the interval $(0, 1)$ is generated. If this number is less than the acceptance probability, then the proposed swap of states is accepted and chains j and k exchange states.

(5) Go back to step 2.

After repeating this process for a sufficiently large number of iterations, the long-run frequencies of states sampled by the *cold chain* are a valid approximation of the posterior distribution.

Successfully swapping states allows a chain that is otherwise stuck on one peak in the landscape of trees to explore other peaks. For example, if the cold chain is stuck on a peak in the posterior distribution of trees, swapping states with another (heated) chain allows the cold chain to jump to another peak in a single cycle. As a result, the cold chain can more easily traverse the space of trees.

The major disadvantage of $(MC)^3$ is its execution time. $(MC)^3$ takes time proportional to the number of chains being run. The number of heated chains needed for adequate mixing in phylogenetic inference problems is likely to be data dependent. We have found that running four coupled chains results in successful convergence for several problems where standard MCMC techniques fail (Huelsenbeck *et al.*, 2001) but it is quite possible that adequate mixing will require many more heated chains for difficult data sets. When dealing with large data sets, running even four chains may be computationally prohibitive using traditional algorithms running on single machines.

PARALLEL (MC)^3

In this paper, we introduce a p(MC)^3 algorithm that significantly decreases the execution time of a Bayesian analysis using (MC)^3. p(MC)^3 works by running Markov chains in parallel. Specifically, p(MC)^3 spreads Markov chains among processes. A process performs all computation associated with its assigned chain(s). This includes calculating the likelihood function, the most computationally intensive operation in any given iteration.

In (MC)^3, all chains proceed to the next iteration in step with each other. Consequently, swaps are made between chains in the same generation. p(MC)^3 breaks away from the notion of all chains proceeding in step with each other. For strict correctness, however, swaps between chains must still take place in the same generation. The exchange rule described below guarantees sequential in-order execution of the swaps.

> **Exchange rule**. Let $c_{i,k}$ be the i-th chain in generation k where $i \in (1, 2, \ldots, n)$ and $k \geq 0$. Chain $c_{i,x}$ can exchange with chain $c_{j,y}$ if and only if $x = y$.

In order to provide results that are identical in both sequential and parallel implementations, p(MC)^3 must strictly adhere to the above exchange rule.

ALGORITHM

Heat swapping

p(MC)^3 invokes communication among processes when a chain exchanges state information with another chain. Since chains may reside on different processes, communication among chains leads to communication among processes. High communication costs can severely degrade the performance of any parallel algorithm.

When using p(MC)^3 with phylogenies, state information can be several megabytes in size. Tree data structures and associated conditional likelihoods account for this large size. Communicating such structures carries significant performance penalties. In both message passing and shared memory implementations of (MC)^3, communication time can exceed computation time if large data structures are being exchanged. Inexpensive communication operations are essential for efficient parallelization of (MC)^3.

p(MC)^3 reduces communication costs by exchanging heats rather than states. That is, chain states and their associated data structures are never communicated. Rather, the heat values associated with each chain are swapped. Since the heat value associated with each chain is unique to that chain, we can effectively swap the identities of two chains by exchanging the heat values. Once heats are swapped, the swapping chains accept new states based on their newly acquired heat values. Using the heat exchange mechanism, there is no need to communicate the chain states and associated data structures, resulting in interprocessor messages of a few bytes, rather than messages several megabytes in size.

Both heat swapping and state swapping adhere to the exchange rule. Heat swapping, however, provides a more efficient way to effect swaps. Furthermore, heat swapping is particularly easy to implement. A swapping chain can readily access its partner's heat value once both have synchronized and communicated swap acceptance information with each other. Once the swap is accepted, heat values can be swapped without further communication.

State swapping code requires two rounds of communication once chains have synchronized. In the first round, swap acceptance information is conveyed to the swap partner. In the second round, the chain state is communicated if the swap has been accepted. Heat swapping, on the other hand, requires only one round of communication. In the first round, swap acceptance information is conveyed to the swap partner. If the swap is accepted, the current heat value is replaced by the heat value received in the swap acceptance information. By sending the random number used in making the swap decision between the processors along with the rest of the swap information, we can ensure that the processors make identical swap acceptance decisions without having to communicate additional information.

Synchronization

In p(MC)^3, heat exchanges call for synchronization. The exchange rule requires that chains synchronize with one another before swapping heats. A swapping chain must receive swap acceptance data from its swap partner before accepting or rejecting a swap. The number of swaps, and thus the amount of synchronization, affects our algorithm's concurrency. We present two exchange schemes that address synchronization in p(MC)^3 with varying degrees of efficiency and ease of use. Both schemes adhere to the exchange rule and therefore provide results that are identical to the sequential version in terms of phylogenetic data (provided that the number of chains and random number seeds are identical).

Global exchange scheme In the global exchange scheme, chains do not interact until an exchange must take place. After *all chains* have completed a specified number of iterations, two randomly selected chains, c_i and c_j, exchange states.

In this scheme, *all chains* must wait for c_i and c_j to complete the exchange. A *barrier* operation (a barrier is a synchronization operation that requires all processes to have entered it before any process is allowed to continue) ensures that swaps are complete before allowing any chain to continue. All chains move collectively from iteration to iteration. The result is that chains are guaranteed to be in the same iteration at the time of a swap, thereby satisfying the exchange rule.

In the message passing version, the global exchange scheme is easy to implement. A barrier operation is performed and

acceptance information is then explicitly communicated via *send* and *receive* operations. The following pseudo-code demonstrates how the message passing version works.

```
For each generation
    Propose a new state
    Accept or reject the new state
    Synchronize using a barrier
    Send/receive swap acceptance info
    If a swap is desirable
       Swap heats
```

In a shared memory implementation of the global exchange scheme, no explicit data communication is required. However, since all data structures are shared, care must be taken to avoid over-writing data that is currently being read by another process [write-after-read (WAR)—a race condition possible when the same two chains swap in succession]. This is accomplished by using two sets of n (where n is the number of chains) swap information holders. With every iteration, a sense variable is reversed in order to alternate between the two sets. The following pseudo-code demonstrates this process (i is the executing chain and j is the chain chosen for swapping).

```
For each generation
    Propose a new state
    Accept or reject the new state
    If sense == 0
       SwapInfoReference
         = sharedSwapInfo_1
    else if sense == 1
       SwapInfoReference
         = sharedSwapInfo_2
    SwapInfoReference[i] = mySwapInfo
    Synchronize using a barrier
    If swapping this iteration AND
       SwapInfoReference[j] is desirable
          Swap heats
    Reverse sense
```

Although the barrier provides a quick and easy implementation, the high cost of communication that results from its use can interfere with obtaining good speed improvements. Communication costs can be especially expensive when using high latency, low bandwidth interconnects.

The major disadvantage of this scheme, however, is that using a barrier can lead to wasted processing time in non-swapping chains due to uneven computation times in each iteration of each chain (the computation time in each iteration of each chain is in part conditional on the type of move, which for MrBayes is randomly chosen). While chains involved in the swap are required to wait to communicate and exchange state acceptance information, uninvolved chains must merely wait for the barrier to complete.

Point-to-point exchange scheme We now present a scheme that minimizes idle time during state exchanges. The scheme works by predetermining which two chains swap in each iteration. Uninvolved chains are allowed to proceed to the next iteration. As a result, little computation time is wasted waiting for swaps to complete.

In this scheme, we exploit the fact that the selection of chains (chain temperatures) for swap attempts in each generation is independent of the swaps that have taken place previously. Thus, we can generate the sequence of swap attempts before the run starts, either using true random numbers or a pseudo-random number generator. Once a sufficiently long sequence of swap attempts has been generated, the swap attempt sequence is shared among all processes. At any given generation, a chain can determine if it is involved in the swap by checking the random number sequence. If a chain is not involved in the swap, there is no need to wait for swapping to complete. That is, the computation in the next generation for a *non-swapping* chain is in no way dependent on the swap. Therefore, chains that are not involved in the swap can begin work on the next generation concurrently. Unlike a chain in the global exchange scheme, a chain in this scheme is independent until the time of a swap and can be several generations ahead of the currently swapping chains.

Synchronization in this scheme is achieved through the use of point-to-point synchronization operations. In the message passing programming model, synchronization and data communication are combined in the send and receive operations. The following pseudo-code demonstrates how the message passing version of the point-to-point exchange scheme works.

```
For each generation
    Propose a new state
    Accept or reject the new state
    If swapping in this iteration
       Send/receive swap acceptance info
       If swap is desirable
          Swap heats
```

When using the heat exchange method, a send operation is used to send swap acceptance information to the swap partner. This includes sending the likelihood and prior probability of the current state as well the chain's heat and swap decision random number.

Point-to-point synchronization in the shared memory model is accomplished through the use of flags. One complication, however, is that swaps between various chains of different generations may be in progress at the same time. In order to avoid race conditions and deadlocks, we use two sets of n^2 flags and swap holders. Each flag uniquely identifies a swapping pair. The two sets ensure that successive swaps between the same chains do not overwrite data in swap holders before it has been read. We alternate between the two sets with every swap of the same chains through the use of n sense

variables (one per pair of processes). The following pseudo-code shows how synchronization is maintained in the shared memory version of the point-to-point exchange scheme.

```
If swapping in this iteration
    If sense[j] == 0
       SwapInfoReference
         = sharedSwapInfo_1
       flagReference = flagSet_1
    else if sense[j] == 1
       SwapInfoReference
         = sharedSwapInfo_2
       flagReference = flagSet_2
    SwapInfoReference[i][j] = mySwapInfo
    Release flagReference[i][j]
    Acquire flagReference[j][i]
    If SwapInfoReference[j][i] is
       desirable Swap heats
    Reverse sense[j]
```

If chains c_i and c_j are swapping, c_i first releases flag $[i, j]$ and then acquires flag $[j, i]$. Chain c_j performs a similar operation by first releasing flag $[j, i]$ and then acquiring flag $[i, j]$.

Data exchange is accomplished through the use of n^2 swap holders, each holding swap acceptance information for swapping chains. Before synchronizing, chains place swap data into the corresponding holder. Once chains have synchronized, a chain can readily access its partner's swap data.

The advantage of the point-to-point exchange scheme is that non-swapping chains no longer idle while a swap is taking place. Moreover, the use of point-to-point synchronization rather than barrier synchronization greatly reduces communication cost and synchronization time.

SYSTEMS AND METHODS

We evaluate p(MC)^3 on a cluster of 8 Compaq AlphaServer 4100 servers running Compaq Unix 4.0F with TruCluster v 1.6 extensions. Each AlphaServer is equipped with four 600 MHz 21164A processors and 2 GB of shared memory. The page size in Compaq Unix is 8192 bytes.

All the servers are connected via an Ethernet and a high-speed Memory Channel network. The Ethernet network supports communication at 100 Megabit/s and is common in most networked environments. The Memory Channel (Gillett, 1996) is a PCI-based crossbar network that provides higher bandwidth and lower latency communication than Ethernet, with a peak point-to-point bandwidth of approximately 83 MB/s and a one-way latency of 3.3 μs for a 64-bit remote-write operation.

We evaluate two implementations of p(MC)^3: a message passing version and a shared memory version. In the message passing version, we experiment with both Ethernet and Memory Channel networks to determine the effects of network bandwidth and latency on the algorithm. We use MPICH 1.2.4 (Gropp *et al*., 1996), a freely available implementation of MPI that works with TCP/IP networks, on the Ethernet network. We use Compaq MPI for evaluating our algorithm with the high-speed Memory Channel network. Compaq MPI is a high performance version of MPICH 1.1.1 specially designed to take advantage of the Memory Channel interface.

The shared memory version was implemented using Cashmere (Stets *et al*., 1997) on the Memory Channel. Cashmere is a software *distributed shared memory* (DSM) system for network multicomputers that provides the illusion of shared memory across a distributed collection of machines.

The p(MC)^3 algorithm and its sequential counterpart (MC)^3 were implemented in C as part of the parallel version of MrBayes 3.0. In order to compile both parallel and sequential implementations of MrBayes, we used gcc version 2.8.1 with the $-$03 flag for optimization.

In our experiments, we use two data sets. The first is a small data set of nine *replicase* sequences from the bacteriophage family Leviviridae (Bollback and Huelsenbeck, 2001). The second is a large data set consisting of 140 ITS sequences of the genus *Astragalus*, and related plant species (Sanderson and Wojciechowski, 2000). Each data set is run under a general time reversible model of DNA substitution (GTR; Tavaré, 1986). In the Leviviridae data set, the number of species in the tree is small. The effect is that the cost of the likelihood function is low as well. An important result of small computation cost is that the communication costs become correspondingly more important. The *Astragalus* data set describes a phylogeny that contains many species, thereby resulting in a costly likelihood function. Communication costs in the large data set are less likely to interfere with speedup since the amount of computation far exceeds communication.

We measure speedup using 1, 2, 4, 8, 16, 24 and 32 chains on as many processors. In order to capture worst-case communication costs, we require that a swap takes place at every generation. In all our experiments, we run p(MC)^3 for 20 000 generations.

RESULTS AND DISCUSSION

Figure 1 presents speedups obtained with both global and point-to-point exchange schemes using shared memory. All speedup figures are based on execution times of the sequential version of MrBayes (Table 1A). Figure 2 shows speedups with Compaq MPI using Memory Channel and with MPICH using a 100 Megabit/s Ethernet network, both using the point-to-point exchange scheme. All figures show speedups for runs of the Leviviridae and *Astragalus* data sets.

The point-to-point exchange scheme achieves nearly linear speedup for both MPI and Cashmere implementations. While message passing has the advantage of allowing application developers to ensure that only required data is communicated

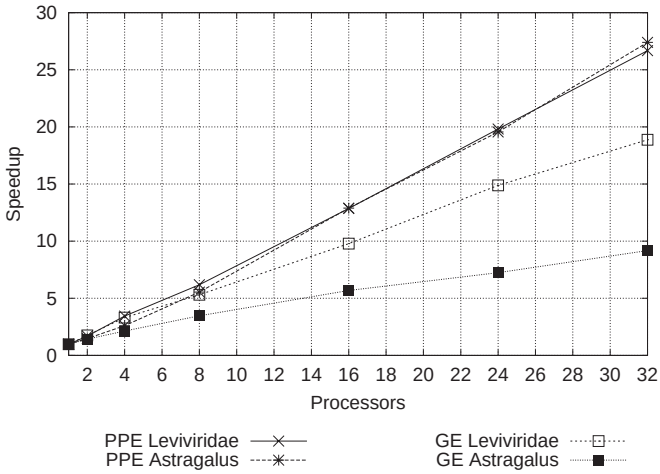

Fig. 1. Speedup of MrBayes using Cashmere and Memory Channel network. Results for global (GE) and point-to-point (PPE) exchange schemes are given.

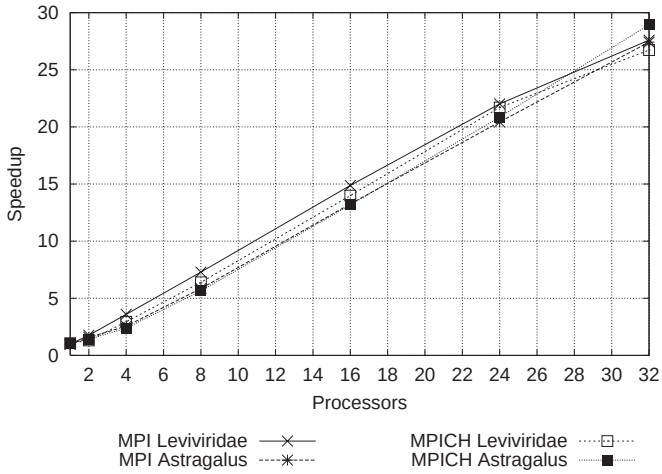

Fig. 2. Speedup of MrBayes using Compaq MPI and MPICH. Results for the point-to-point exchange scheme are given.

Table 1. Speedup of MrBayes using Cashmere

Chains	A	B	
	Seq.	GE	PPE
1	18:103	0:0	0:0
2	37:201	9.4:25.3	9.2:24.3
4	74:389	14.1:42.6	9.4:33.3
8	150:871	24.9:56.9	14.1:31.6
16	305:1871	32.5:68.8	16.2:30.1
24	466:2790	35.8:72.9	18.0:28.5
32	622:3881	36.3:74.7	18.5:28.3

(A) Sequential execution times in seconds of MrBayes on a Compaq Alpha 4100. (B) Percentage wait times at synchronization points for Cashmere using the global (GE) and point-to-point (PPE) exchange schemes. Times for Leviviridae and *Astragalus* in both sub-figures are separated by a colon in each cell.

across nodes (resulting in a 3% improvement in performance over Cashmere for Leviviridae), shared memory has the advantage of more effectively utilizing the hardware support for shared memory within each node [as evidenced by the 2 and 4% better performance of Cashmere at 4 processors for the two data sets; performance differences are small since $p(MC)^3$ incurs low communication overhead]. Systems such as Cashmere allow expansion of the use of shared memory across nodes without much loss in performance.

On Cashmere, the point-to-point exchange scheme achieves a three-fold performance increase over the global exchange scheme. This is a result of low cost point-to-point synchronization and high concurrency of computation and communication. Table 1B provides the percentage of total running time that each processor on average spends waiting at synchronization points for both point-to-point and global exchange schemes. Wait time in the point-to-point exchange scheme with 32 processors and the *Astragalus* analyses is 28.3%. Wait time in the global exchange scheme for the same configuration, on the other hand, is 74.7%. The lower wait times of the point-to-point exchange scheme are due in part to the fact that only two processors synchronize no matter how many processors there may be. The effect is that the amount of synchronization is not a function of the number of processors. The wait times also reflect the fact that non-swapping chains can begin new computations while synchronization is taking place. That is, more time is spent performing useful computation than waiting at synchronization points. As speedups for the scheme indicate, low synchronization costs coupled with overlapping communication and computation provides high scalability.

Low synchronization costs and high concurrency, however, do not fully account for the good speedups of the point-to-point exchange scheme. Specifically, a 28.3% wait time on 32 processors with *Astragalus* should actually result in a speedup of 21× not 27×. The remaining performance improvement can be attributed to caching effects. With our data sets, each chain touches a minimum 1 MB of data when performing the likelihood calculation. When running more than eight chains on the sequential implementation of $p(MC)^3$, chains must compete with each other for the 8 M of on-board cache. In the parallel implementation, however, no competition is necessary since we run only one chain per processor. The result is that tree and conditional likelihood structures of all chains stay in cache, thereby increasing the speed of likelihood calculations significantly.

The high wait times of the global exchange scheme are a result of two factors—the communication cost of barrier synchronization and the computational imbalance among the chains. The communication cost of barrier synchronization

is proportional to the number of nodes. Most importantly, however, the wait times reflect the imbalance in the computation performed in each chain. In MrBayes, each chain randomly selects a move (i.e. proposal mechanism) every iteration. Since moves vary from chain to chain (e.g. some chains select a computationally expensive move while others select an inexpensive move), the time taken by each chain to arrive at the barrier varies as well. The barrier prevents any chain from continuing to the next iteration until all chains have reported in. As a result, all chains must wait for the slowest processor (i.e. the one that selected the most expensive move), thereby wasting valuable computation time. The amount of wasted computation time increases with the number of chains and data size (with larger data sizes, some moves become more expensive). Consequently, the global exchange scheme suffers from poor scalability.

As results from the MPI runs indicate, both Leviviridae and *Astragalus* achieve good speedups when using either the Ethernet or the Memory Channel network. The high performance of Leviviridae on the Ethernet network is particularly remarkable since a small data set tends to generate a high number of messages per second. We believe that the exceptional performance of the Ethernet implementation is a result of two factors: the low-cost communication benefits of the heat exchange method and the performance advantages afforded by our experimental platform. The heat exchange method reduces the amount of communication to a degree that the algorithm is less dependent on the speed of the interconnect. This can be seen in the speedup graphs where speedup increases steadily even when communicating across nodes. Another factor is that communication overhead on our Compaq Alpha is minimal when exchanging data between two processors on the same node. Communication is costly only when communicating across nodes. Since each node has four processors, a swap may not always require making use of the network.

With the point-to-point exchange scheme, the total amount of data transmitted is independent of the size of the input data set and the number of chains (recall that only two chains communicate no matter how many chains there may be). For our runs on MPI, a maximum of 1.28 MB may have to be communicated with 20 000 generations of $(MC)^3$, assuming an attempted swap in each cycle.

In comparing results from the two data sets, we note that the speedups for the two data sets are comparable, with the Leviviridae analyses showing better speedups at small numbers of processors. The *Astragalus* analyses' higher computation to communication ratio might suggest that it should perform better. However, the Leviviridae analyses had smaller waiting times (measured as a percentage of the total analysis time) than the *Astragalus* analyses (Table 1B). The large computation size of the *Astragalus* data increases the potential worst case computational imbalance among chains, which is the major contributor to the worst-case wait time. The chain initiating a swap must wait for its partner to finish its evaluation. Moreover, this penalty propagates to other chains that depend on the swapping chain. This larger wait time offsets the increased computation to communication ratio for *Astragalus*. In Leviviridae, the imbalance or skew in the computational load of each chain is smaller since the size of computation is small. That is, a chain need not wait for a long period of time even in the worst case. However, as the speedup graphs demonstrate, the performance of the *Astragalus* analyses scales better with larger numbers of processors (results at 32 processors, the maximum number in our experimental platform, show how the performance of the two data sets begin to deviate, with the *Astragalus* data set continuing to show near linear performance improvement). The degradation in performance for the smaller data set with the increased number of chains is due to the increased waiting times (cf. Table 1B). As the number of chains increases, the number of swaps per chain decreases. As a result, the variance in the number of swap attempts among chains increases, leading to longer wait times. The smaller data set is more susceptible to this effect because of the high communication to computation ratio. For the larger data set, as the number of swaps per chain decreases, the variation in computational burden among processors decreases, leading to shorter waiting times for this data set as the number of processors increases (cf. Table 1B).

CONCLUSION

In this study, we have developed a new $p(MC)^3$ algorithm. The goals of the parallel algorithm are to attain a high level of concurrency and to minimize communication costs. By taking advantage of the random nature of $(MC)^3$, our algorithm allows non-swapping chains to proceed to the next generation. The effect is that chains perform useful computation for a majority of the execution. Moreover, our algorithm swaps heats rather than states, thereby decreasing communication overhead significantly.

The parallel algorithm has been implemented in both message passing and shared memory programming models. The performance of both implementations has been analyzed and found to achieve near optimal speedup. By experimenting with both Ethernet and the Memory Channel networks, we have found that our algorithm scales well to a large number of processors. High scalability opens up the possibility of running a large number of chains for better mixing. Finally, we found that our algorithm performs equally well on both small and large data sets. Thus, a parallel $(MC)^3$ analysis need not be limited to special data sets.

We are currently exploring load balancing techniques that would result in idle chains being detected and put to useful work, in addition to the parallelization of the likelihood calculation itself. By making sure that all processors have work to do throughout the computation, we hope to achieve better

speed improvements and utilize larger numbers of processors efficiently.

ACKNOWLEDGMENTS

We thank the University of Rochester Systems Group and Umit Rencuzogullari for their helpful insights into parallel $(MC)^3$. S.D. was supported for this work in part by NSF grants CCR-9702466, CCR-9988361, CCR-0219848, ECS-0225413, and EIA-0080124; by the US Department of Energy Office of Inertial Confinement Fusion under Cooperative Agreement No. DE-FC03-92SF19460; and by an equipment grant from Compaq. J.P.H. was supported by NSF grants DEB-0075406 and MCB-0075404 and by a Guggenheim Fellowship. F.R. was supported by Swedish Research Council grant 621-2001-2963.

REFERENCES

Bollback,J.P. and Huelsenbeck,J.P. (2001) Phylogeny, genome evolution, and host specificity of single-stranded RNA bacteriophage (Family Leviviridae). *J. Mol. Evolution*, **52**, 117–128.

Felsenstein,J. (1968) Statistical inference and the estimation of phylogenies. Ph.D. dissertation, University of Chicago.

Geyer,C.J. (1991) Markov chain Monte Carlo maximum likelihood. In Keramidas (ed.), *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*. Interface Foundation, Fairfax Station, pp. 156–163.

Gilks,W.R. and Roberts,G.O. (1996) Strategies for improving MCMC. In Gilks,W.R., Richardson,S. and Spiegelhalter (eds) *Markov chain Monte Carlo in Practice*. Chapman & Hall, London, 89–114.

Gillett,R. (1996) Memory channel: an optimized cluster interconnect. *IEEE Micro*, **16**, 12–18.

Green,P.J. (1995) Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, **82**, 711–732.

Gropp,W., Lusk,E., Doss,N. and Skjellum,A. (1996) A high-performance, portable implementation of the MPI message passing interface standard. *Technical Report ANL/MCS-TM-213*, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

Hastings,W.K. (1970) Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, **57**, 97–109.

Huelsenbeck,J.P. and Ronquist,F. (2001) MrBayes: Bayesian inference of phylogenetic trees. *Bioinformatics* **17**, 754–755.

Huelsenbeck,J.P., Ronquist,F., Nielsen,R. and Bollback,J.P. (2001) Bayesian inference of phylogeny and its impact on evolutionary biology. *Science*, **294**, 2310–2314.

Larget,B. and Simon,D. (1999) Markov chain Monte Carlo algorithms for the Bayesian analysis of phylogenetic trees. *Mol. Biol. Evolution*, **16**, 750–759.

Li,S. (1996) Phylogenetic tree construction using Markov chain Monte carlo. Ph.D. dissertation, Ohio State University, Columbus.

Maddison,D.R. (1991) The discovery and importance of multiple islands of most parsimonious trees. *Syst. Zool.*, **40**, 315–328.

Maddison,D.R., Swofford,D.L. and Maddison,W.P. (1997) NEXUS: an extensible file format for systematic information. *Syst. Biol.*, **46**, 590–621.

Mau,B. (1996) Bayesian phylogenetic inference via Markov chain Monte carlo methods. Ph.D. dissertation, University of Wisconsin, Madison.

Mau,B. and Newton,M. (1997) Phylogenetic inference for binary data on dendrograms using Markov chain Monte Carlo. *J. Comp. Graph. Stat.*, **6**, 122–131.

Mau,B., Newton,M. and Larget,B. (1999) Bayesian phylogenetic inference via Markov chain Monte carlo methods. *Biometrics*, **55**, 1–12.

Metropolis,N., Rosenbluth,A.W., Rosenbluth,M.N., Teller,A.H. and Teller,E. (1953) Equations of state calculations by fast computing machines. *J. Chem. Phys.*, **21**, 1087–1091.

MPIF (Message Passing Interface Forum). (1994) MPI: a Message-Passing Interface standard. *Int. J. Supercomput. Appl.*, **8**, 157–416 (Special Issue).

Newton,M.A., Mau,B. and Larget,B. (1999) Markov chain Monte Carlo for the Bayesian analysis of evolutionary trees from aligned molecular sequences. In F. Seillier-Moseiwitch (ed.), *Statistics in Molecular Biology and Genetics*. IMS Lecture Notes-Monograph Series, **33**, 143–162.

Rannala,B. and Yang,Z. (1996) Probability distribution of molecular evolutionary trees: a new method of phylogenetic inference. *J. Mol. Evolution*, **43**, 304–311.

Ronquist,F. and Huelsenbeck,J.P. (2003) Mr Bayes 3: Bayesian phylogenetic inference under mixed models, *Bioinformatics*, **19**, 1572–1574.

Salter,L.A. and Pearl,D.K. (2001) Stochastic search strategy for estimation of maximum likelihood phylogenetic trees. *Syst. Biol.*, **50**, 7–17.

Sanderson,M.J. and Wojciechowski,M.F. (2000) Improved bootstrap confidence limits in large-scale phylogenies, with an example from Neo-Astragalus (Leguminosae). *Syst. Biol.*, **49**, 671–685.

Stets,R., Dwarkadas,S., Hardavellas,N., Hunt,G., Kontothanassis,L., Parthasarathy,S. and Scott,M.L. (1997) Cashmere-2L: software coherent shared memory on a clustered remote-write network. *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pp. 170–183.

Tierney,L. (1994) Markov chains for exploring posterior distributions (with discussion). *Ann. Stat.*, **22**, 1701–1762.

Yang,Z. (1994) Estimating the pattern of nucleotide substitution. *J. Mol. Evolution*, **39**, 105–111.

Yang,Z. and Rannala,B. (1997) Bayesian phylogenetic inference using DNA sequences: a Markov chain Monte carlo method. *Mol. Biol. Evolution*, **14**, 717–724.