BIOINFORMATICS ORIGINAL PAPER

Vol. 27 no. 9 2011, pages 1255–1261
doi:10.1093/bioinformatics/btr140

Phylogenetics

Advance Access publication March 16, 2011

MrBayes on a Graphics Processing Unit

Jianfu Zhou[1], Xiaoguang Liu[1,*], Douglas S. Stones[1,2,3], Qiang Xie[4] and Gang Wang[1]

[1]Nankai-Baidu Joint Laboratory, College of Information Technical Science, Nankai University, 300071 Tianjin, China, [2]School of Mathematical Sciences, Monash University, VIC 3800 Australia, [3]Clayton School of Information Technology, Monash University, VIC 3800 Australia and [4]Institute of Entomology, College of Life Sciences, Nankai University, 300071 Tianjin, China

Associate Editor: David Posada

ABSTRACT

Motivation: Bayesian phylogenetic inference can be used to propose a 'tree of life' for a collection of species whose DNA sequences are known. While there are many packages available that implement Bayesian phylogenetic inference, such as the popular MrBayes, running these programs poses significant computational challenges. Parallelized versions of the Metropolis coupled Markov chain Monte Carlo (MC^3) algorithm in MrBayes have been presented that can run on various platforms, such as a graphics processing unit (GPU). The GPU has been used as a cost-effective means for computational research in many fields. However, until now, some limitations have prevented the GPU from being used to run MrBayes MC^3 effectively.

Results: We give an appraisal of the possibility of realistically implementing MrBayes MC^3 in parallel on an ordinary four-core desktop computer with a GPU. An earlier proposed algorithm for running MrBayes MC^3 in parallel on a GPU has some significant drawbacks (e.g. too much CPU–GPU communication) which we resolve. We implement these improvements on the NVIDIA GeForce GTX 480 as most other GPUs are unsuitable for running MrBayes MC^3 due to a range of reasons, such as having insufficient support for double precision floating-point arithmetic. Experiments indicate that run-time can be decreased by a factor of up to 5.4 by adding a single GPU (versus state-of-the-art multicore parallel algorithms). We can also achieve a speedup (versus serial MrBayes MC^3) of more than 40 on a sufficiently large dataset using two GPUs.

Availability: GPU MrBayes (i.e. the proposed implementation of MrBayes MC^3 for the GPU) is available from http://mrbayes-gpu.sourceforge.net/.

Contact: liuxg74@yahoo.com.cn

Supplementary information: Supplementary data are avaliable at *Bioinformatics* online.

Received on September 17, 2010 ; revised on March 6, 2011; accepted on March 10, 2011

1 INTRODUCTION

Given the DNA sequences of several organisms, Bayesian inference can be used to infer their phylogenetic 'tree of life'. MrBayes (Huelsenbeck and Ronquist, 2001) is a popular software package that implements the Metropolis coupled Markov chain Monte Carlo (MC^3) sampling method for Bayesian inference of phylogeny.

> Metropolis coupled MCMC is ideally suited to implementation on a parallel processing machine or even on a network of workstations (each processor being assigned one chain), since each chain will in general require about the same amount of computation per iteration, and interactions between chains are simple. (Gilks *et al.*, 1996)

In fact, several modified versions of MrBayes MC^3 have been developed that allow the user to run multiple chains in parallel on multi-CPU-based hardware (cf. Section 2.1). The purpose of this article is to instead analyze the possibility of implementing MrBayes MC^3 in parallel on a graphics processing unit (GPU). Note that, in the rest of this article, whenever we discuss MrBayes, we implicitly refer to version 3.1.2 (the current version).

1.1 MrBayes MC^3 overview

MrBayes MC^3 is capable of Bayesian inference of phylogeny in a range of situations. In this article, we exclusively focus on the 4-by-4 nucleotide substitution GTR+Γ and GTR+I+Γ models, which has the settings

```
nucmodel = 4by4
nst = 6
```

and either `rates = gamma` or `rates = invgamma` (these are set with the `lset` command), and the format setting:

```
datatype = dna
```

We will now give a quick overview of this computation, a detailed description of which can be found in Huelsenbeck and Ronquist (2005).

MrBayes MC^3 runs H Markov chains, each containing a proposed phylogenetic tree, which we denote $\psi_1, \psi_2, \ldots, \psi_H$. In each iteration, we randomly perturb each tree to give another list of trees $\psi'_1, \psi'_2, \ldots, \psi'_H$ and for each Markov chain i decide whether or not to replace ψ_i with ψ'_i. Afterwards, we decide whether or not to swap the states of two chains j and k, where j and k are chosen at random.

By default, MrBayes runs two independent analyses of the same data, which helps the user determine when an accurate tree is found. Each analysis runs $H=4$ Markov chains, three heated chains and one 'cold' chain, which allows for more rapid mixing (Ronquist *et al.*, 2005).

For each chain, a series of computations are performed in the order depicted in Figure 1. The gray box indicates what is computed at a given stage, and the arrow indicates what information is required at a given stage. We let Q denote the (instantaneous) *rate matrix* and X denote the DNA sequence data of the taxa. Further, we let

*To whom correspondence should be addressed.

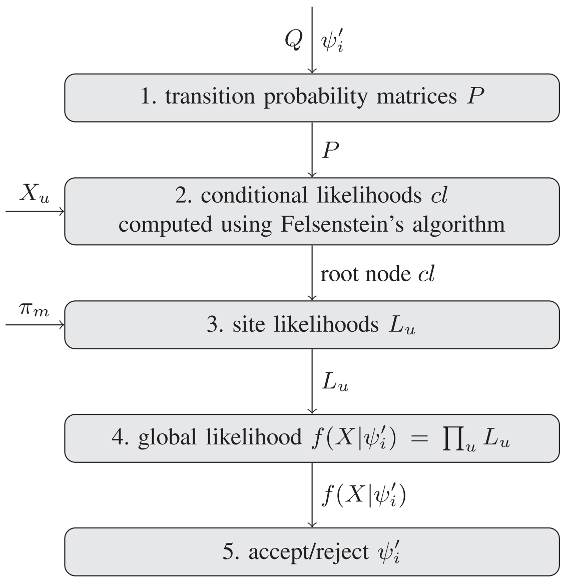
Q
ψ_i'
1. transition probability matrices P
P
X_u
2. conditional likelihoods cl computed using Felsenstein's algorithm
root node cl
π_m
3. site likelihoods L_u
L_u
4. global likelihood $f(X|\psi_i') = \prod_u L_u$
$f(X|\psi_i')$
5. accept/reject ψ_i'

Fig. 1. Flowchart of MrBayes MC^3 in serial.

X_u denote the DNA sequence data at site u and π_m denote the base frequencies of nucleotide $m \in \{A,C,G,T\}$. We compute the conditional likelihoods using the recursive algorithm presented in Felsenstein (1981).

The whole procedure is performed repeatedly, until after many generations (i.e. iterations) the cycle is broken. More generations imply a more reliable result.

1.2 GPU overview

Modern GPUs have a large number of cores, and a single GPU can run tens of thousands of threads concurrently. Despite GPUs being originally designed for high-performance graphics processing, due to being powerful and relatively inexpensive, GPUs have started to be used in a vast range of scientific applications (Owens *et al.*, 2005). Moreover, a GPU can be conveniently added to an existing desktop computer, just as a typical graphics card.

The NVIDIA Corporation introduced Compute Unified Device Architecture (CUDA) (NVIDIA, 2009) which enables developers to write programs for the NVIDIA GPU using a minimally extended version of C language. It is arguably the most important reason for the prosperity of general purpose computing on GPUs.

The GPU-side function of a CUDA program is called a *kernel*, which is executed by numerous GPU threads concurrently. The GPU threads executing the same kernel form one or more GPU *thread blocks*.

The NVIDIA GPU has several different memory types with different behaviors. One is called *global memory*, which is the largest memory available but is also the slowest. All GPU threads have access to the same global memory. To transfer data between CPU and GPU, we must use global memory. Another memory type important for this article is *shared memory*, which is smaller but faster than global memory. Each GPU thread block has its own shared memory accessible to all threads of the block. The memory access procedure is, therefore, an important factor in the efficiency of a GPU-based algorithm. In particular, frequent global memory access will incur a performance drag.

For this article, we exclusively use the NVIDIA GeForce GTX 480, which is priced at US$449.99 (amazon.com, December 2010). We have been quite selective in choosing this particular GPU for several reasons. For example (i) many GPUs are incompatible with CUDA, (ii) many GPUs are unsuitable for running MrBayes MC^3 due to inadequate ability to work with double precision floating-point arithmetic and (iii) some GPUs are simply too expensive.

2 PARALLEL MRBAYES MC^3

A range of algorithms for implementing MrBayes MC^3 in parallel have been proposed which we will discuss in this section.

2.1 Multi-CPU-based architecture

Altekar *et al.* (2004), presented a basic parallel algorithm for MrBayes MC^3; they assigned multiple Markov chains as evenly as possible among multiple CPU cores. They called their algorithm pMC^3 (and we will follow this style of nomenclature in this article). Overall, the default number of Markov chains run by MrBayes is eight (four chains per analysis), but pMC^3 offers no way to run eight Markov chains on more than eight cores. Feng *et al.* (2003, 2006), proposed a parallel version of the MC^3 method which allows any number of Markov chains to be run in parallel on any number of cores. These modifications were subsequently made for MrBayes MC^3 in Zhou *et al.* (2010), which we call hMC^3. Van der Wath *et al.* (2008), presented a version of MrBayes MC^3 for networked computers, such as over the internet.

2.2 GPU architecture

A parallel version of MrBayes MC^3 was proposed in (Pratas and Sousa, 2009; Pratas *et al.*, 2009) and implemented on a range of platforms including the GPU; we call this program gMC^3, which we have obtained via private communication with the authors.

> However, the largest overhead for the GPU systems is the transfer of the data to and from the graphics card. This results in a large penalty in the execution time to such an extent that for the 8800GT its execution time is at the end larger than the baseline. (Pratas *et al.*, 2009)

The most time-consuming component of MrBayes MC^3 is the evaluation of the conditional likelihoods *cl* (see also the Supplementary Material). The single goal of gMC^3 is to distribute the computation of *cl* among GPU threads, ignoring all other influencing factors. When the CPU–GPU communication overhead is taken into account, the algorithm's overall performance is poor.

The aim of this article is, therefore, to resolve the problems with gMC^3 and consider the prospect of running MrBayes MC^3 on a GPU in a realistic setting.

2.3 Other GPU-based phylogenetics

It is, however, already possible to run other packages for Bayesian phylogenetic inference on a GPU. Charalambous *et al.*, 2005, implemented Randomized Axelerated Maximum Likelihood (RAxML) (Stamatakis *et al.*, 2005) on a GPU, although recent RAxML development seems to be focused on SSE3 technology.

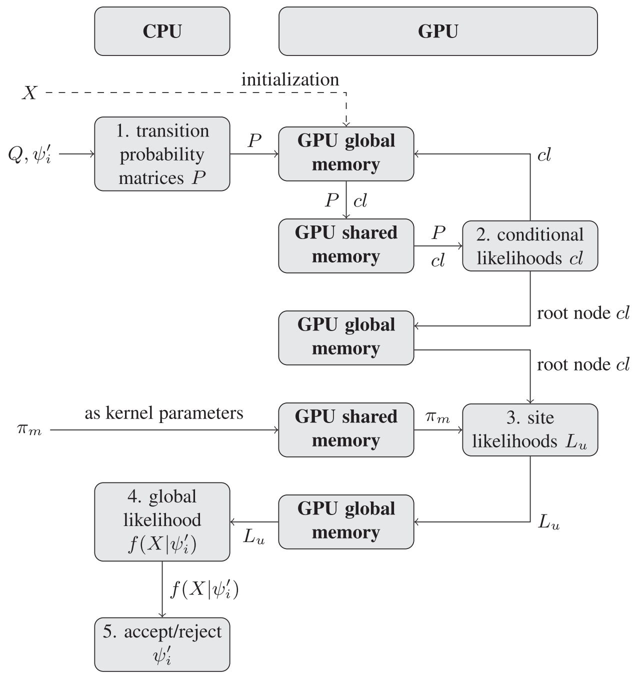

Fig. 2. Flowchart of nMC^3.

Suchard and Rambaut, 2009, implemented a library, called beagle-lib, for evaluating phylogenetic likelihoods on GPUs, which provides an open API for a range of phylogenetic softwares. It has recently been reported that future versions of MrBayes will encompass the beagle-lib, thus enabling GPU computing. However, it is still a work in progress, so we do not present a comparison with MrBayes+beagle-lib in this article.

3 AN IMPROVED ALGORITHM FOR THE GPU

3.1 Overview

We will now offer some simple, but significant, improvements on the earlier proposed gMC^3 algorithm on the GPU. We call this improved algorithm nMC^3 and we depict the concept in Figure 2 for the *i*-th chain. Algorithm 1 is a pseudo-code description of one CPU process in nMC^3. In Algorithm 1, we also label where Stages 1–5 (as in Fig. 2) occur.

We will now list the key advantages of nMC^3 over gMC^3.

- Alongside the GPU, the CPU may also perform computations in parallel, distributing chains among CPU processes. In gMC^3, it is possible to distribute some of the conditional likelihood calculations to the CPU, although the authors of gMC^3 particularly focused on the GPU-side computation.
- We employ pipelining in order to reduce the idle time of both the CPU and GPU, and to 'overlap' CPU–GPU communication with computation (see Section 3.2.2 for more details).
- The results in Stages 1 and 3 are each transferred as a single batch. Hence, in one iteration of nMC^3, only two CPU–GPU data transfers are required, whereas gMC^3 performs four CPU–GPU data transfers at each non-leaf node.

Algorithm 1 nMC^3: one CPU process

```
1.  for all assigned Markov chains $i$ do // Stage 1
2.      Propose a new tree $\psi_i'$ by perturbing $\psi_i$
3.      for all non-root nodes $k$ do
4.          for all discrete rates $r$ do
5.              Compute transition probability matrix $P = P(\psi_i', k, r)$
6.          end for
7.      end for
8.      Transfer $P$ from CPU to GPU
9.  end for
10. for all assigned Markov chains $i$ do // Stages 2–3
11.     for all non-leaf nodes $k$ do
12.         Call kernel to compute conditional likelihoods $cl = cl(\psi_i', k, u, r, n)$ for all sites $u$, discrete rates $r$, and nucleotides $n$ using Felsenstein's algorithm
13.     end for
14.     Call kernel to compute site likelihoods $L_u$ for all sites $u$
15.     Transfer $L_u$ from GPU to CPU
16. end for
17. for all assigned Markov chains $i$ do // Stages 4–5
18.     Synchronize with corresponding GPU stream
19.     Compute $f(X|\psi_i') = \prod_u L_u$
20.     Accept/reject $\psi_i'$
21. end for
```

- Stage 3, computing the site likelihoods, is now performed in parallel by the GPU also. While Stage 3 is a simple computation, performing it on the GPU side means that we can postpone the GPU to CPU data transfer until after Stage 3, when less data are required to be transferred.

We also note that, while the results of nMC^3 and MrBayes MC^3 will be similar provided enough generations are performed, they will not necessarily be identical.

3.2 The nitty-gritty

The aim of this section is to give additional details of nMC^3, expanding on the overview given in Section 3.1.

3.2.1 Parallelism overview The nMC^3 algorithm parallelizes MrBayes MC^3 at two levels. First, we distribute multiple Markov chains among multiple processes on the CPU side, using a message passing programming model.

Secondly, on the GPU side, the task of computing a single *cl* is assigned to a single GPU thread. Within a thread block, we include the computation of *cl* for all nucleotides *n*, discrete rates *r* and as many sites *u* as possible.

Moreover, we also compute L_u on the GPU side using a similar task assignment as for *cl*, the purpose of which is to reduce the CPU–GPU communication overhead (cf. Section 3.2.3). The management of the GPU threads for computing *cl* and L_u is performed automatically by the GPU.

3.2.2 Pipelining In nMC^3, we employ *pipelining*, where each Markov chain is treated as a *workflow* as depicted in Figure 2. A workflow consists of one CPU process and one GPU stream.

Since we can only efficiently run one CPU process per available CPU core, CPU processes are formed from a collection of *assigned*

Markov chains. From a workflow's perspective, it shares a single CPU process with other workflows (if necessary). For example, one such arrangement is depicted below, where 8 Markov chains $1,2,\ldots,8$ are used and 4 CPU cores a,b,c,d are available. Algorithm 1 describes a single CPU process in nMC^3.

$$\begin{array}{c} \text{Markov chain} \leftrightarrow \text{workflow} \\ \begin{array}{|c|c|c|c|} \hline \underbrace{1,2}_{\text{a}} & \underbrace{3,4}_{\text{b}} & \underbrace{5,6}_{\text{c}} & \underbrace{7,8}_{\text{d}} \\ \hline \end{array} \\ \text{CPU process} \leftrightarrow \text{CPU core} \end{array}$$

The advantages are that (i) multiple workflows can perform their respective CPU-side tasks in parallel, (ii) among workflows assigned to the same CPU process, when a workflow is required to perform its GPU-side tasks, another workflow may continue to use the available CPU core and (iii) within a workflow, the CPU process and GPU stream can be run at the same time.

Combined with NVIDIA's new generation Fermi architecture (employed by the GeForce GTX 480), it is therefore possible to simultaneously perform

(1) parallel execution of CPU processes,

(2) parallel execution of GPU streams (i.e. concurrent kernel execution) and

(3) data transfers between CPU and GPU,

thereby improving concurrency and overlapping CPU–GPU communication with computation.

3.2.3 CPU–GPU communication In each iteration of each Markov chain i, on the newly proposed phylogenetic tree ψ_i', for non-leaf node k, site u, discrete rate r and nucleotide $n \in \{A,C,G,T\}$, we are required to compute a conditional likelihood $cl_n = cl(\psi_i', k, u, r, n)$ using Felsenstein's recursive algorithm, which requires knowledge of the corresponding transition probability matrices P^L and P^R and conditional likelihoods cl_b^L and cl_b^R, where $b \in \{A,C,G,T\}$, from the two child nodes of k. In nMC^3, in each iteration of each chain, on the new tree:

- we first compute the transition probability matrices P for each non-root node, then use only one data transfer to upload all of the P to the GPU global memory before commencing Stage 2.
- The cl of the leaf nodes are determined by the DNA sequence data, which we transfer to GPU global memory during initialization.
- Subsequently, we call a kernel to compute the site likelihoods L_u on the GPU side. The base frequencies π_m are uploaded to the GPU as kernel parameters. Then in each workflow, we transfer the results to the CPU side as a single batch.

For gMC^3, to compute $cl = cl(\psi_i', k, u, r, n)$ over all u, r and n at a non-leaf node k of tree ψ_i', (i) the required P^L and P^R are computed on the CPU side and transferred together to the GPU side (utilizing linear memory), (ii) two data transfers retrieve all of the required cl^L and cl^R, respectively, from the CPU side and (iii) the newly computed cl are returned back to the CPU side.

We approximate the number and size of transfers for a single generation in Table 1. However, we wish to caution the reader that this table is intended as a rough guide aimed to highlight the differences between nMC^3 and gMC^3; it does not take into account every factor (e.g. initialization, local update, kernel calls, etc.).

Table 1. Estimating CPU–GPU transfers in nMC^3 and gMC^3

	nMC^3		gMC^3	
	Size per transfer	Number	Size per transfer	Number
$P \rightarrow$ GPU	$R(2N-2)d(P)$	H	$2Rd(P)$	$H(N-1)$
$cl \rightarrow$ GPU	–	–	$4RMd(cl)$	$2H(N-1)$
$cl \rightarrow$ CPU	–	–	$4RMd(cl)$	$H(N-1)$
$L_u \rightarrow$ CPU	$Md(L)$	H	–	–

Key (for rooted phylogenetic trees).

N,	number of taxa;
$2N-2$,	number of non-root nodes (or branches, or edges);
$N-1$,	number of non-leaf nodes;
M,	number of sites (i.e. the length of the DNA sequences);
H,	number of Markov chains per analysis ($H=4$ by default in MrBayes);
R,	number of discrete rates ($R=4$ by default in MrBayes);
$d(x)$,	space used to store x in memory. In particular, $d(P)$, $d(cl)$, and $d(L)$ denote the space used to store a transition probability matrix, a conditional likelihood, and a site likelihood, respectively. In MrBayes, $d(P) = (4\times 4)\times 4$ bytes $= 64$ bytes, $d(cl) = 4$ bytes and $d(L) = 8$ bytes.

3.2.4 Global memory versus shared memory A standard technique in GPU programming is to transfer repeatedly used data to shared memory, thus reducing the total number of global memory accesses.

- For fixed i, k and r, we can reuse P^L and P^R, for the computation of $cl = cl(\psi_i', k, u, r, n)$ over various n and u.
- For fixed i, k, u and r, we can reuse cl_b^L and cl_b^R for $b \in \{A,C,G,T\}$, for the computation of cl_n over various n.

With this in mind, inside a GPU kernel, nMC^3 first transfers the required P^L, P^R, cl_b^L and cl_b^R from global memory to shared memory, then accesses them from shared memory whenever required for some computation. This technique was also employed in gMC^3. Furthermore, in nMC^3, a similar technique is also used during the computation of site likelihoods L_u, when the base frequencies π_m are repeatedly used.

3.2.5 Synchronization By *synchronization*, we refer to when some component of a program is paused and waits for some dependent computations or transfers before continuing. In nMC^3:

- *Within a workflow.* The CPU process is instructed not to begin Stage 4 until its corresponding GPU stream has returned the required L_u.
- *Between two workflows.* After Stage 5, two chains are chosen at random to swap their states (in practice, only the heat values of both chains need to be swapped). For this to occur, we only need to synchronize the two CPU processes where the two chains reside, respectively, while the remaining CPU processes can

Table 2. Dataset information

Dataset	No. of taxa	DNA length	No. of generations	Run-time (s)				
				MrBayes MC^3	hMC^3/pMC^3	gMC^3 with 1 GPU	nMC^3 with 1 GPU	nMC^3 with 2 GPUs
1	26	1546	10 00 000	4 238	1 227	10 034	1343	731
2	37	2238	10 00 000	14 962	4 363	20 654	1784	943
3	111	1506	5 00 000	17 234	5 352	20 321	1765	929
4	234	1790	1 00 000	15 160	4 275	10 015	797	414
5	288	3386	1 00 000	31 700	10 456	19151	–	750

continue. Of course, if the two chosen chains are assigned to the same CPU process, there is no need to synchronize.

In gMC^3, however, the CPU and GPU synchronize whenever there is a data transfer between them.

4 EXPERIMENTS AND DISCUSSION

In this section, we will give experimental results that indicate the performance and scalability of nMC^3 and compare it with the algorithms gMC^3, pMC^{3}and hMC^3. Some additional performance tests can be found in the Supplementary Material.

4.1 Architecture

We implement nMC^3 using a modified version of pMC^3 for MrBayes version 3.1.2. Experiments for nMC^3 are performed in two settings: once with one GPU and once with two GPUs. In these experiments, we always run eight Markov chains (four chains per analysis), while all other settings in MrBayes are set to their defaults. The platform details are listed below.

Operating System	Red Hat Enterprise Linux 5
CPU	AMD Phenom II X4 945 (4 cores)
Memory	2 × 2 GB
GPU	NVIDIA GeForce GTX 480
GPU Memory	1.5 GB
Graphics Driver	NVIDIA Device Driver, Version 195.36.15

GCC Version 4.1.2 with the -O3 and -Wall flags was used for compiling pMC^3 and hMC^3, and also the CPU-side codes of nMC^3 and gMC^3. The GPU-side codes of nMC^3 and gMC^3 were compiled using CUDA Toolkit Version 2.3. For pMC^3, hMC^{3}and nMC^3, MPICH2 Version 1.1 was used for message passing.

4.2 Datasets

For testing, we use several datasets that were used by the fourth author in phylogenetic research. Dataset 1 is a group of 18S rDNA from 26 representative species belonging to 13 different families of Trichophora (Insecta: Hemiptera: Heteroptera), which was used to study the evolutionary relationships among the main lineages of Eutrichophora (Xie *et al.*, 2005). Dataset 2 is a group of 18S rDNA from representatives of all groups at order level in Euhemiptera, which includes 33 family-level taxa, which was used to investigate the effect of rDNA length variation on alignment and phylogenetic reconstruction among the suborders of Hemiptera (Xie *et al.*, 2008). Dataset 3 is a group of metazoan 18S rDNA which includes 111 taxa at class or order level and was used to reconstruct the phylogenetic relationships between the phyla of metazoans. Dataset 4 is a group of eukaryotic 18S rDNA which includes 234 taxa at class or order level and was used to reconstruct the phylogenetic relationships between the kingdoms of eukaryotes. Dataset 5 is a group of 23S–28S rDNA from cellular organisms which includes 288 taxa at class or order level and was used to study the positional homology of nuleotides according to length variation in the secondary structure of corresponding rRNA. The results of Datasets 3, 4 and 5 are yet to be published. All five datasets are available for download with the implementation of nMC^3. Table 2 lists some of the basic details of datasets 1–5.

4.3 Results

4.3.1 Run-time Included in Table 2 is the time taken to analyze the datasets on the platform described in Section 4.1 with (i) serial MrBayes MC^3 using 1 CPU process, (ii) either pMC^3 or hMC^3 (whichever is faster) using 4 CPU processes, (iii) gMC^3 using one GPU and one CPU process and (iv) nMC^3 using either one or two GPUs and 4 CPU processes. We run each experiment only once, since the difference in run-times between repeated experiments is negligible (at most 1%).

4.3.2 Speedup We make the following definitions:

- The *speedup* is the number of times faster nMC^3 using four CPU processes and either one or two GPUs performs when compared with MrBayes MC^3 run in serial using a single CPU process.
- The 4-*speedup* is the number of times faster nMC^3 using four CPU processes and either one or two GPUs performs versus the fastest algorithm using four CPU processes (either pMC^3 or hMC^3).
- The g-*speedup* is the number of times faster nMC^3 using four CPU processes and either one or two GPUs performs versus gMC^3 using one CPU process and one GPU.

Predictably, when using two GPUs the algorithm runs almost twice as fast as when using a single GPU. Table 3 lists the various speedups, computed from the data in Table 2.

Note that, we are unable to run dataset 5 using nMC^3 with 4 CPU processes and a single GPU due to insufficient GPU global memory.

4.3.3 Scalability We will now give the experimental results that indicate the scalability of nMC^3, gMC^3, pMC^{3}and hMC^3. Figures 3

Table 3. Speedup of nMC^3

Dataset	Speedup		4-Speedup		g-Speedup	
	1 GPU	2 GPUs	1 GPU	2 GPUs	1 GPU	2 GPUs
1	3.2	5.8	0.9	1.7	7.5	13.7
2	8.4	15.9	2.4	4.6	11.6	21.9
3	9.8	18.6	3.0	5.8	11.5	21.9
4	19.0	36.6	5.4	10.3	12.6	24.2
5	–	42.3	–	13.9	–	25.5

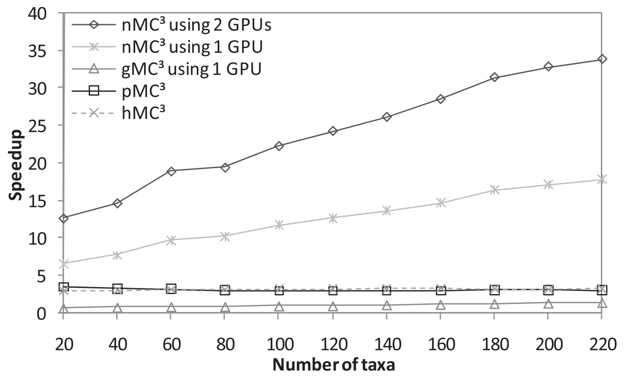

Fig. 3. Speedup of nMC^3, gMC^3, pMC^{3}and hMC^3 on the last a taxa in dataset 4, where $a \in \{20,40,\ldots,220\}$.

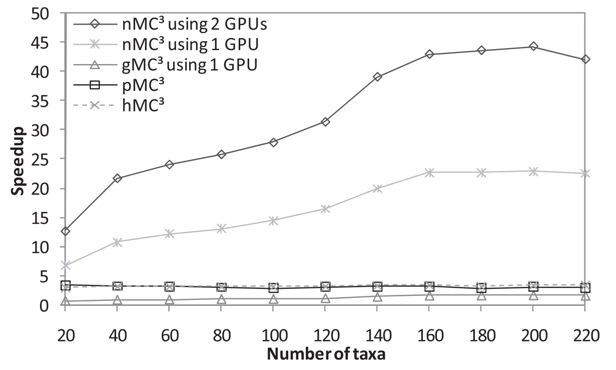

Fig. 4. Speedup of nMC^3, gMC^3, pMC^3, and hMC^3 on the last a taxa in dataset 5, where $a \in \{20,40,\ldots,220\}$.

and 4, respectively, plot the speedup on the dataset consisting of the last a taxa of either Dataset 4 or Dataset 5, as a varies. Figure 5 plots the speedup on a group of simulated datasets of 60 taxa, which were generated with Seq-Gen version 1.3.2 (Rambaut and Grassly, 1997), consisting of b sites, as b varies. In each case, we run nMC^3, gMC^3, pMC^3, or hMC^3 for 10000 generations. Furthermore, all the simulated datasets are also available for download with the implementation of nMC^3.

4.4 Discussion

Figures 3–5 suggest that nMC^3 achieves increasing speedup as the problem size increases, whereas gMC^3, pMC^{3}and hMC^3 all have approximately constant speedups.

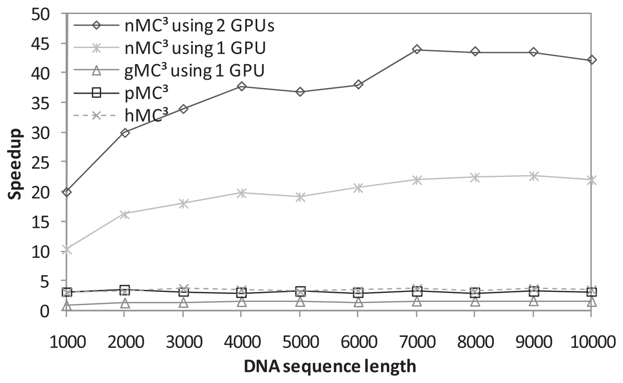

Fig. 5. Speedup of nMC^3, gMC^3, pMC^{3}and hMC^3 on a group of simulated datasets of 60 taxa, consisting of b sites, where $b \in \{1000,2000,\ldots,10000\}$.

Two important factors that affect the performance of a parallel algorithm are *communication overhead*, where processes require data from other processes, and *load imbalance*, where one process takes significantly longer than another. As the problem size increases, we attribute these near-constant speedups to load imbalance for pMC^3, and both load imbalance and communication overhead in hMC^3. For gMC^3 , communication overhead increases with the problem size, as does the time required for the component that is run in serial.

5 CONCLUSION

We have analyzed the prospect of realistically implementing a GPU-based version of MrBayes MC^3 on an ordinary computer. For this research, we exclusively focused on the 4-by-4 nucleotide substitution GTR+Γ and GTR+I+Γ models, and used the NVIDIA GeForce GTX 480. We tested the algorithm on datasets of interest to biologists, along with a group of simulated datasets. The experiments suggest that a single GPU can improve the performance of MrBayes MC^3 by up to a factor of roughly 19 (versus serial MrBayes MC^3). Moreover, we can increase this speedup further by using additional GPU(s).

- It is now possible to use a GPU to substantially reduce the run-time of MrBayes MC^3.
- For MrBayes MC^3, achieving a speedup on a GPU will, for an appropriately sized dataset, be less expensive than achieving the equivalent speedup using multi-CPU-based hardware. Moreover, a GPU can be conveniently added to an existing desktop computer.

As time progresses, we anticipate that the GPU will offer increasingly greater potential for Bayesian phylogenetic inference. However, we encountered several obstacles in this research that the reader should be aware of.

- Currently, very few GPUs are suitable for running the proposed algorithm due to inadequate support for double precision floating-point arithmetic. NVIDIA's Fermi architecture is a recent technology that provides sufficient support for double precision floating-point arithmetic.
- There is a restriction on the largest dataset that can be processed by a single GPU due to memory capacity. It is possible to

lessen this restriction by running the proposed algorithm with additional GPU(s) or on fewer CPU processes.

- Only NVIDIA GPUs support CUDA, which enables general purpose applications to run on GPUs.

Aside from the above, we implement the proposed algorithm only on Linux, although the source code is available for download and modification.

ACKNOWLEDGEMENTS

We would like to thank the authors of Pratas *et al.* (2009) for supplying us with gMC^3 and providing some helpful comments; Wenjun Bu for his assistance; Charles Semple and Gang Hu for valuable feedback; and the anonymous referees for their independent testing of the proposed algorithm and their constructive criticism.

Funding: National High Technology Research and Development Program of China (2008AA01Z401), NSFC of China (60903028,61070014), Science and Technology Development Plan of Tianjin (08JCYBJC13000), and Key Projects in the Tianjin Science and Technology Pillar Program. Stones was supported in part by an Australian Research Council discovery grant.

Conflict of Interest: none declared.

REFERENCES

Altekar,G. *et al.* (2004) Parallel Metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference. *Bioinformatics*, **20**, 407–415.

Charalambous,M. *et al.* (2005) Initial experiences porting a bioinformatics application to a graphics processor. In *Advances in Informatics: Proceedings of PCI 2005*, Springer, Berlin/Heidelberg, pp. 415–425.

Felsenstein,J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, **17**, 368–376.

Feng,X. *et al.* (2003) Parallel algorithms for Bayesian phylogenetic inference. *J. Parallel Distrib. Comput.*, **63**, 707–718.

Feng,X. *et al.* (2006) PBPI: a high performance implementation of Bayesian phylogenetic inference. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing, IEEE Computer Society*, Los Alamitos, CA, USA, p. 40.

Gilks,W.R. *et al.* (1996) *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC, Boca Raton, FL, USA.

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

Huelsenbeck,J.P. and Ronquist,F. (2005) Bayesian analysis of molecular evolution using MrBayes. In *Statistical Methods in Molecular Evolution*, Springer, New York, pp. 183–226.

NVIDIA Corporation (2009) *NVIDIA CUDA Programming Guide Version 2.3.1*. NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, CA.

Owens,J.D. *et al.* (2005) A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, Vol. 13, pp. 21–51.

Pratas,F. and Sousa,L. (2009) Applying the stream-based computing model to design hardware accelerators: a case study. In *Embedded Computer Systems: Architectures, Modeling, and Simulation, Proceedings of SAMOS 2009*. Vol. 5657 of *Lecture Notes in Computer Science*, Springer, Berlin/Heidelberg, pp. 237–246.

Pratas,F. *et al.* (2009) Fine-grain parallelism using multi-core, cell/BE, and GPU systems: accelerating the phylogenetic likelihood function. In *2009 International Conference on Parallel Processing*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 9–17.

Rambaut,A. and Grassly,N.C. (1997) Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Comp. App. Biosci.*, **13**, 235–238.

Ronquist,F. *et al.* (2005) *MrBayes 3.1 Manual*. Tallahassee, FL: School of Computational Science, Florida State University. Available at: <http://mrbayes.csit.fsu.edu/mb3.1_manual.pdf> (last accessed date March 26, 2011).

Stamatakis,A. *et al.* (2005) RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, **21**, 456–463.

Suchard,M.A. and Rambaut,A. (2009) Many-core algorithms for statistical phylogenetics. *Bioinformatics*, **25**, 1370–1376.

Van der Wath,R.C. *et al.* (2008) Bayesian phylogeny on grid. In *Bioinformatics Research and Development*, Vol. 13, pp. 404–416.

Xie,Q. *et al.* (2005) The Bayesian phylogenetic analysis of the 18S rRNA sequences from the main lineages of Trichophora (Insecta: Heteroptera: pentatomomorpha). *Mol. Phylogenet. Evol.*, **34**, 448–451.

Xie,Q. *et al.* (2008) 18S rRNA hyper-elongation and the phylogeny of Euhemiptera (Insecta: Hemiptera). *Mol. Phylogenet. Evol.*, **47**, 463–471.

Yang,Z. (1994) Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods. *J. Mol. Evol.*,**39**, 306–314.

Zhou,J. *et al.* (2010) A new hybrid parallel algorithm for MrBayes. In *ICA3PP (1), Vol. 6081 of Lecture Notes in Computer Science*, Springer, Berlin/Heidelberg, pp. 102–112.