2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Optimizing the Bayesian Inference of Phylogeny on Graphic Processors

Cheng Ling
Tsinghua University and Guangzhou Institute of Advanced Technology, Chinese Academy of Science
s0897918@gmail.com

Chunbao Zhou
Supercomputing Center
Computer Network Information Center
Chinese Academy of Science
zhoucd@sccas.cn

Arong Luo
Key Laboratory of Zoological Systematics and Evolution
Institute of Zoology
Chinese Academy of Science
luoar@ioz.ac.cn

Guoguang Zhao
Institute for Network Science and Cybespace
Tsinghua University
zhaogg@bioinfo.ac.cn

Tsuyoshi Hamada
Department of Computer and Information Science
Nagasaki University
hamada@nacc.nagasaki-u.ac.jp

Xiaoyan Zhu
Department of Computer Science and Technology
Tsinghua University
zxy-dcs@tsinghua.edu.cn

Abstract

Searching for the evolutionary relationships between groups of organism has become a routine procedure in molecular biology. MrBayes is a popular model based phylogenetic inference tool using Bayesian statistics. Unfortunately, the computational cost is very high, resulting in undesirably long execution time. In this paper, we present what we believe the fastest solution of the MrBayes MC^3 algorithm running on off-the-shelf graphic processors. The performance benefits are offered by the multi-granularity parallelism model, coarse-grained GPU kernel system, efficient thread arrangement strategy and GPU code level optimizations. MrBayes $goMC^3$ (proposed herein) provides a significant performance improvement over the sequential MrBayes MC^3 by a speedup of up to 48× when using single Tesla C2075 GPU card, whereas a speedup factor of 77× can be achieved when using dual GPUs. In comparison to the state-of-the-art version of other publicly available GPU implementations of MrBayes MC^3, the cumulative optimizations adopted in $goMC^3$ resulted in a speedup of up 2.5× over oMC^3 (v1.0), 1.75× over $tgMC^3$ (v1.0) and 1.46× over nMC^3(v2.1.1) for realistic empirical biological datasets. Besides, experimental results indicated that $goMC^3$ outstrips these GPU implementations on the analysis of simulated datasets composed of ultra-large-scale sequences. As a consequence, the reported performance improvement of $goMC^3$ is significant and appears to scale well with increasing dataset sizes.

Keywords CUDA, phylogenetic analysis, MrBayes

1. Introduction

Phylogenetic analysis is used widely for the study of molecular sequence variation. Among the major algorithms for phylogenetic inference, Maximum Likelihood [1] and Bayesian Inference [2] approaches are often favored as they can infer the relationships of sequences by explicit evolutionary models. MrBayes [3] is a popular computer package for the Bayesian inference of phylogeny, which further employs the Metropolis Coupled Markov Chain Monte Carlo (MC^3) algorithm to converge independent likelihood estimation around a stable value, allowing the output of a robust consensus tree for given set of species. However, model based assessment of phylogenetic analysis adds to the computational burden of likelihood estimation, posing significant computational challenges. Nonetheless, available for over 10 years, MrBayes has been cited more than 14,000 times. The level of usage suggests that any improvement in the execution speed of MrBayes MC^3 will result in significant impact in phylogeny. Since the implementation of each individual chain in MrBayes MC^3 algorithm is independent, several parallelized versions of the MC^3 algorithm [4, 5, 6, 7] employed multiple CPU processes to compute these chains in parallel, while the potential parallelism of MrBayes MC^3 can be further exploited by massive amount of threads. Since GPUs are becoming increasingly important as powerful, efficient and low cost option for general purpose parallel computations, research efforts for high performance phylogenetic analysis have relied not only on traditional multi-core processors, but also on many-core GPUs substantially. To the best of our knowledge, the first GPU-based parallel version of MrBayes MC^3 is the gMC^3 method [8], which gained additional performance improvement by parallelizing the computation of conditional likelihood probability (*clp*). Furthermore, a series of CPU-GPU heterogeneous computing based methods, namely nMC^3 [9], aMC^3 [10],

978-1-4799-8006-2/15 $31.00 © 2015 IEEE
DOI 10.1109/CCGrid.2015.13

tgMC^3 [11] and oMC^3 [12] were reported in literatures. Among these attempts, different contributions were taken to speed up the algorithm.

This paper presents what we believe the fastest solution of MrBayes MC^3 running on off-the-shelf graphic cards so far. The proposed goMC^3 method further exploited the multi-threaded parallel computing capability of CUDA GPUs and reduced the data transfer overhead between heterogeneous computing platforms by cumulative sets of optimizations. In comparison to the state-of-the-art version of other publicly available GPU implementations of MrBayes MC^3, the goMC^3 method achieved the best performance.

The rest of the paper is organized as follow. The following section will first present the background on the MrBayes MC^3 algorithm, together with an overview of the characteristics of target architecture. Section 3 will then report the detailed design and implementation of the MrBayes MC^3 algorithm on the above platform. After that, comparative implementation results are presented in Section 4. An evaluation of the proposed implementation is presented in Section 5 before final conclusions are drawn.

2. Background

The essential process of MrBayes by proposing new tree states normally needs to be repeated many millions of times, with the most time-consuming operation in each replicate being the likelihood computation of the phylogeny. The following subsection will first present theoretical background on the MrBayes MC^3 and then its potential parallelizable modules, followed by an architectural overview of the target hardware platform.

2.1 Overview of MrBayes MC^3

MrBayes is a software tool for the Bayesian inference of phylogeny, which employs the standard algorithm of Markov Chain Mote Carlo (MC^2) to approximate the posterior probabilities of phylogenetic trees, with the Bayes' theorem, prior probabilities, stochastic starting-states, calculations of acceptance probability and others. Practically, MC^3, a variant of MC^2, is usually adopted in MrBayes, which runs *n* chains with *n-1* as heated chains and the other as the only cold chain. The heated chains are incrementally heated by certain parameters, while the cold chain could swap states with the heated chains and so can be potentially across deep valleys in the landscape of trees. Only statistical information from the cold chain is used for phylogenetic inference. The core calculations in MrBayes are recursively performed by generating new tree states based on topological rearrangements on old tree states, updating transition probabilities, and then calculating likelihoods of new tree states to decide whether the rearrangements are accepted. Various evolutionary models are employed in MrBayes, e.g. JC69 [13], HKY85 [14], GTR [15].

The likelihood calculation method of a phylogeny can be easily present by using a particular tree case illustrated in Figure 1, while the general expression for the likelihood estimation of a phylogeny can be draw from that case later. Let τ denote the tree topology, *1, 2, ..., i* denote the node labels of the tree, v_i denote the

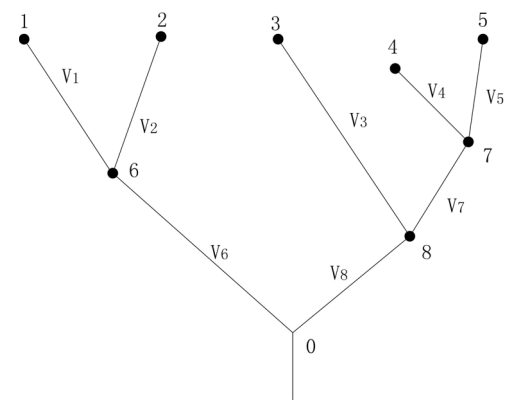

Figure 1. The tree used in the discussion of the likelihood computation.

segment length between the node *i* and its ancestral node, S_i denote the state of node *i*, the likelihood probability at site *x* of the given tree, input sequence data and evolutionary model would be the product of the probabilities of change in each tree segment:

$$L(x \mid \tau, v, \theta) = P_{s_0 s_6}(v_6, \theta) P_{s_6 s_1}(v_1, \theta) P_{s_6 s_2}(v_2, \theta) \times P_{s_0 s_8}(v_8, \theta) P_{s_8 s_3}(v_3, \theta) P_{s_8 s_7}(v_7, \theta) P_{s_7 s_4}(v_4, \theta) P_{s_7 s_5}(v_5, \theta); \quad (1)$$

Where *P* stands for the transition probabilities between two states, and θ contains the parameters of the evolutionary model for calculating *P*. Hence, *P (v, θ)* stands for the transition probabilities over a segment length of *v* under specified evolutionary parameters θ. For nucleotide sequences, the state of internal node would belong to one of the four states, *A, C, G, T*, and there is only one state per site for tip nodes, which can be observed from input sequences. Therefore, the likelihood for site *x* will be the sum of the four assignments as present in (2).

$$L(x \mid \tau, v, \theta) = \sum_{s_0=A}^{T} \sum_{s_6=A}^{T} \sum_{s_7=A}^{T} \sum_{s_8=A}^{T} P_{s_0 s_6}(v_6, \theta) P_{s_6 s_1}(v_1, \theta) P_{s_6 s_2}(v_2, \theta) \times P_{s_0 s_8}(v_8, \theta) P_{s_8 s_3}(v_3, \theta) P_{s_8 s_7}(v_7, \theta) P_{s_7 s_4}(v_4, \theta) P_{s_7 s_5}(v_5, \theta); \quad (2)$$

The summation is over all 256 ways that nucleotide can be assigned to the internal nodes of the tree, and the value increase exponentially with the number of internal nodes. To solve the problem, Felsenstein [1] realized a considerable economy by transferring it to (3), where the pattern of parentheses reflects the relationships of the topology of the tree. The pruning algorithm enabled the calculation of likelihood can be evaluated by working outwards from the innermost parentheses, which can be simply understood to start the likelihood calculations from the tips of the tree and moving downward one by one.

$$L(x \mid \tau, v, \theta) = \sum_{s_0=A}^{T} \left\{ \sum_{s_6=A}^{T} P_{s_0 s_6}(v_6, \theta) \left[P_{s_6 s_1}(v_1, \theta)\right] \left[P_{s_6 s_2}(v_1, \theta)\right] \right\} \times$$

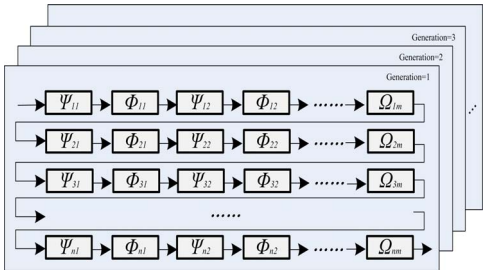

Figure 2. The implementation architecture of MrBayes MC^3..

$$\left\{\begin{array}{c}\sum_{s_8=A}^{T} P_{s_0 s_8}(v,\theta)\left[P_{s_8 s_3}(v_3,\theta)\right]\times \\ \left[\sum_{s_7=A}^{T} P_{s_8 s_7}(v_7,\theta)\left(P_{s_7 s_4}(v_4,\theta)\right)\left(P_{s_7 s_5}(v_5,\theta)\right)\right]\end{array}\right\};$$

(3)

For the likelihood computation of individual node k, if let $\sigma_i(k)$ and $\sigma_j(k)$ denote its immediate descendants, the conditional likelihood probabilities *clp* can be calculated by (4).

$$L_{s_k}(x \mid \tau, v, \theta) = \left(\sum_{s_{\sigma i(k)}} P_{s_k s_{\sigma i(k)}}\left(v_{\sigma i(k)},\theta\right) L_{s_{\sigma i(k)}}\right)\left(\sum_{s_{\sigma j(k)}} P_{s_k s_{\sigma j(k)}}\left(v_{\sigma j(k)},\theta\right) L_{s_{\sigma j(k)}}\right);$$

(4)

Let π denote the prior probabilities of the four base residues, the overall likelihood of the tree for site x under consideration is then obtained by (5).

$$L(x \mid \tau, v, \theta) = \sum_{s_k=A}^{T} \pi_{s_k} L_{s_k}\,; (5)$$

Normally, the evolutionary rate among different sites is assumed to be drawn from a gamma distribution [16]. If continuous gamma-distributed rate variation is assumed, than the likelihood probability of the tree becomes:

$$L(x \mid \tau, v, \theta, \beta) = \int_0^{\infty} \sum_{s_k=A}^{T} \pi_{s_k} L_{s_k}^{r} f(r|\beta) dr\,; (6)$$

Where $f(r|\beta)$ is the density of the rate r under the gamma model, β is the vector holds the parameters of the gamma distribution. Typically, the integral is impossible to calculate. In [17], Yang et al. proposed an approach by applying G discrete gamma distribution categories to approximate the calculation, thus the likelihood of node k then becomes:

$$L(x \mid \tau, v, \theta, \beta) = \sum_{n=1}^{K} \sum_{s_k=A}^{T} \pi_{s_k} L_{s_k}^{n} \frac{1}{G}; (7)$$

It is possible to improve the accuracy of likelihood computations by increasing the G value. By default, MrBayes sets the number of gamma category to 4 and we leave the setting in our experiments, as it is sufficient in most cases. For the purpose of a standardized evaluation against other implementations, we used MrBayes v3.1.2 (unless stated otherwise) under a commonly applied general time reversible model allowing inv-gamma distributed variation in the rate of substitution over sites and a proportion of invariable sites. A detailed description of MrBayes MC^3 settings can be found elsewhere [18].

Table 1. The possible computation complexity of tree nodes.

Φ		Ψ			
β_1	**$\beta_{2,3}$**	**down_0**	**down_1**	**down_2**	**down_3**
0	0	γ_0	γ_1	γ_2	γ_3
1	0	$\gamma_0+\beta_1$	$\gamma_1+\beta_1$	$\gamma_2+\beta_1$	$\gamma_3+\beta_1$
0	1	$\gamma_0+\beta_2$	$\gamma_1+\beta_2$	$\gamma_2+\beta_2$	$\gamma_3+\beta_2$
1	1	$\gamma_0+\beta_1+\beta_2$	$\gamma_1+\beta_1+\beta_2$	$\gamma_2+\beta_1+\beta_2$	$\gamma_3+\beta_1+\beta_2$

2.2 Analysis of potential parallelizable modules

The following symbols are defined to denote the modules implementing the likelihood calculation:

- Ψ: calculate the likelihood of internal nodes;
- Φ: scale the likelihood;
- Ω: calculate the likelihood of a phylogeny;

In the core calculation of MrBayes MC^3, the internal nodes of each tree are traversed iteratively in post-order manner for likelihood calculation. It is more useful to present the sorted internal nodes of each tree as a sequence of elements and the whole computational tasks as a 3-D matrix M_{ijz}, where i, j are the chain and tree node number ($1 \leqq i \leqq n$) ($1 \leqq j \leqq m$), z is the number of the generations in MC^3. All elements in the sequence are restricted to be sorted in the same row from left to right and all sequences are arranged in increasing length order from top to bottom. For each row, the modules of Ψ and Φ are executed alternately, and module Ω is finally implemented to produce the likelihood of a phylogeny. The implementation architecture of MrBayes MC^3 thus can be briefly illustrated in Figure 2.

The process of proposing new tree states normally needs to be repeated many millions of times in the MC^3 algorithm, and there is always a statistical method be taken to move the evolutionary parameters of the current tree in each generation of MC^3. In general, moving tree branch length and topology are the two operations that harnessed frequently. The former updates transition probability matrix and the latter changes the relationships of nodes. Hence, two crucial steps need to be performed in module Ψ, which are the load of transition probabilities and the calculation of likelihood probabilities, denoted as step α_1 and α_2. For step α_2, MrBayes MC^3 firstly assesses the types of descendent nodes of arbitrary internal node, and labels the node as one of *down_0*, *down_1*, *down_2* and *down_3*. The implementation of step α_2 is then distributed to relevant pass. In addition to Ψ, three steps are supposed to be performed in module Φ, which are the deletion of old scales, the calculation of new scales and the addition of the scales, denoted as β_1, β_2 and β_3, respectively. The steps of β_2 and β_3 can be deemed as one step, as they are normally performed successively. The likelihood calculation may not require a complete calculation including all these steps due to phylogenetic analysis methods, i.e. no need to scale a node sometimes. Hence, the granularity of codes that implements module Ψ and Φ varies between different tree nodes. Let γ_0, γ_1, γ_2, γ_3 denote the complexity of step α_2 implemented in different pass, θ_1 and θ_2 denote the complexity of β_1,

β_2&β_3, the 16 possible granularities are shown in Table 1. Note here the granularity of each case does not include the complexity for α_1, as it is a constant in all cases.

Since these modules can be well parallelized, and the execution time takes over 98% of the total execution time, MrBayes MC^3 is considered suited to be implemented on GPUs as the potential peak performance is significantly high. Nevertheless, the implementation architecture is not straightforward and requires a huge effort to design as well as at implementation level.

2.3 GPU architecture characteristics

GPUs are massively parallel platforms designed to accelerate computationally intensive applications by operating in a Single Instruction Multiple Data (SIMD) model. Multiple threads in the same warp run on identical Stream Multiprocessor (SM), execute the same instructions in parallel, but operate on different data. Threads that performed in parallel on GPUs are grouped into so-called blocks, while blocks are grouped into so-called grids. The dimension of blocks and grids decides the total number of executing threads, which are determined by users before kernel launches. Once a kernel launched, the GPU scheduler assign the execution of each thread block to a specific GPU SM. Since each SM has a limited number of processors, the maximum number of threads that can physically be executed simultaneously is actually fixed, despite that thousands of threads can be launched concurrently. The threads that physically executed in parallel are called a warp of threads. A thread block may contain multiple warps, and the GPU scheduler decides in which order and when to execute each warp. This gives the capability to the scheduler to increase the overall utilization of the GPU SMs by implementing different warps once a warp is ready to perform calculations, while others are waiting for data from long latency global memory access.

The Degree of Parallelism (DOP) is maximized when all threads in a warp agree on their execution path. If threads of a warp diverge via a data-dependent conditional branch, the warp serially executes each branch path taken, disabling threads that are not on that path. When all paths complete, the threads converge back to the same execution path. Extensive thread divergence can have a detrimental effect on performance.

There are different types of memory on GPUs, each with unique functionality, size and transfer speed. Threads that are owned by the same block can share data using fast on-chip memory (shared memory). Perform recursive calculations by using the cached data in shared memory, instead of fetching them from global memory will significantly increase the overall performance. Threads in different blocks can only share data using relatively slow on-board memory (global memory) on a GPU card. The global memory access pattern of the threads can affect substantially the data transfer efficiency, the more coalesced the memory accessing within a half warp threads, the higher the achieved efficiency. Registers of a GPU SM are shared between its processors, and each thread uses an exclusive set of registers. In addition, there are constant memory, texture memory and local memory, which were not found useful in the implementation of the proposed method. The specifications of the targeted GPUs are introduced in section 4.

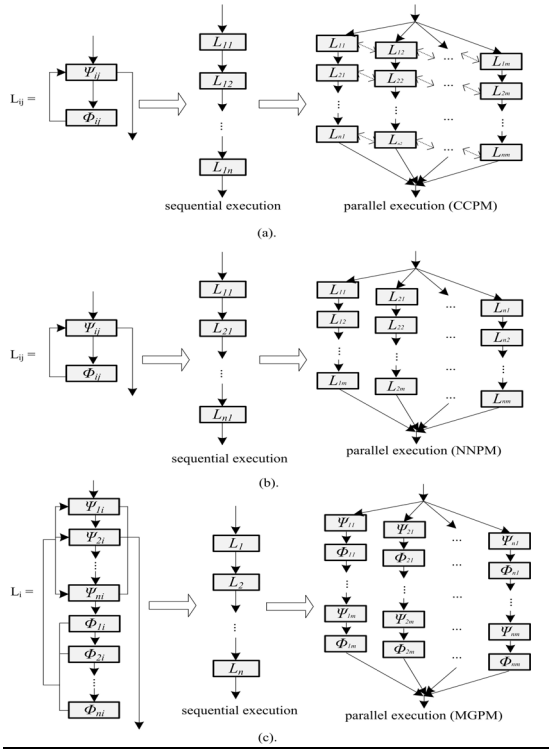

Figure 3. The three different parallelism methods: (a). CCPM; (b) NNPM; (c) MGPM.

3. Our approach

3.1 Multiple granularity parallelism

In this section we present the optimizations on the implementation architecture to exploit parallelism, while the detailed parallel computing method for each module described above will be introduced in section 3.3. Figure 3 shows that the computation flow graphs of three sequential loops as well as the graphs after parallelization have been employed for all three types of parallelism.

Let L_{ij} denote the task that performs the likelihood calculation of node j on the tree sampled by chain i, the left graph in Figure 3(a) then denotes the data dependences of module Ψ and Φ as well as the indeterministic dependences of input data from another node when executing L_{ij}. The middle graph illustrates the computation flow graph of sequential tasks L_{1j} ($1 \leqq j \leqq m$), which are implemented for the likelihood of nodes from chain 1. Based on this implementation architecture, Zhou et al. [9] presented a chain-by-chain parallelization model (CCPM) by assigning the tasks of each chain to relevant GPU streams. During the execution of each stream, all L_{ij} tasks belonging to the chain are continuously implemented by asynchronously launched GPU kernels. As illustrated in the right graph of Figure 3(a), each layer of parallelization is composed of tasks from the same chain, and all tasks are arranged in increasing number order of chains from top to bottom.

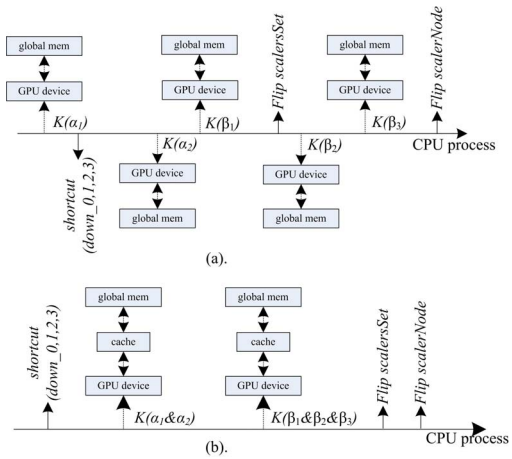

Figure 4. The comparison of two GPU kernel systems: (a) fine-grained kernel system; (b) coarse-grained kernel system.

The parallelization model adopted is coarse-grained in each parallelization layer, as it is possible that there are dependences existing between code regions. For instance, L_{12} for node 2 depends on the results produced by L_{11} if node 2 is the ancestral node of node 1. We use dotted line to highlight the indeterminism between tasks. As a consequence, the DOP of CCPM is limited by the reason that the tasks in parallel may have dependences.

To increase the granularity parallelism, Bao et al. [10] extend the work by proposing a finer granularity node-by-node parallel model (NNPM). In spite of being data dependent upon the previous node in the same chain, nodes from different chains are essentially data independent. As illustrated in the middle graph of Figure 3(b), the computation flow graph of sequential tasks is no longer composed of tasks from the same chain, but the same nodes (depicted by node index) from different chains. By implementing the calculation of each individual node from different streams in parallel, the original program semantics still can be preserved, which has been comparison checked for the parallelization to be correct. As illustrated in the right graph of Figure 3(b), each layer of parallelization is composed of tasks from different chains, rather than the nodes from the same chain, hence there are no data dependence among these tasks. NNPM defines a more effective cooperation scheme between L_{ij} tasks, achieving finer granularity parallelism. However, the degree of parallelism in each layer of parallelization is hard to be identical, as the size of granularity varies dynamically (see Table 1). For instance, it is common that the number of steps is various in L_{11} and L_{21}, despite the two tasks can be executed independently. Due to the unequal amount of parallelizable operations within the tasks on the same parallelization layer, there is a reduction on performance.

The degree of parallelism can be further optimized by exploiting the inner dependency of L_{ij} tasks. By comparing the sequential and parallel computation flow graphs, we derive the sets of dependencies between modules and propose a multiple granularity parallelism model (MGMP) (see Figure 3(c)). To further maximize the DOP, we optimize the implementation architecture by splitting each L_{ij} task into finer grained residues. As illustrated in the left graph of Figure 3(c), the improved model classifies all Ψ and Φ from the same batch of tasks into two groups. The computation flow graph of sequential tasks is a number of L_i, each of which is composed of two groups of sub tasks, namely $g(\Psi_i)$ and $g(\Phi_i)$, $(1 \leqq i \leqq m)$. The right graph of Figure 3(c) shows the parallel execution of MrBayes MC^3 by harnessing multiple granularities. Since both the external and internal data dependences of parallelizable tasks are rearranged, the proposed MGPM improves the DOP, and achieves better performance.

3.2 Coarse-grained GPU kernel systems

Last section presents three parallelism models that one by one improving the DOP of MrBayes MC^3. In this section we present the optimized method of organizing GPU kernels to implement these steps described above. Since a complete likelihood calculation of a tree node contains five steps: α_1, α_2, β_1, β_2 and β_3, the GPU computation of conditional likelihood probabilities (*clp*) under the approach adopted in oMC^3 and nMC^3 is as follows. We assume that the transition probabilities (*tip*) of descendent nodes have already been transferred to GPU global memory. As illustrated in Figure 4(a), the implementation architecture of computing the *clp* of a node can be grouped into five kernel functions named as $K(\alpha_1)$, $K(\alpha_2)$, $K(\beta_1)$, $K(\beta_2)$ and $K(\beta_3)$. These kernels are launched to the GPU device asynchronously (arrows with dotted line), and a few operations are performed by CPU process between each launch (arrows with full line). The essential function of each kernel is described as:

$K(\alpha_1)$ - Transform the *tip* matrix of internal nodes and pad the transformed matrix to *prelike* matrix with new *tip* for ambiguity data, and load the *prelike* matrix to GPU global memory. Note that not all internal nodes need to perform the procedure, as some of them may not have terminal nodes, in which case, load the *tip* matrix to the GPU global memory for $K(\alpha_2)$ is enough.

$K(\alpha_2)$ - Load *tip* or *prelike* matrix from GPU global memory to shared memory, in addition to terminal states and *clp* of its descendents, and distribute the computation of *clp* among GPU threads. The computed *clp* values need to be saved in the global memory as other internal nodes from the same chain may need them.

$K(\beta_1)$ - If the parameter *scalerSet* of the node is activated in MrBayes MC^3, the *lnscaler* value of each site belonging to the phylogeny should be subtracted by the old scales belonging to the node. In the mean time, the CPU process flips the *scalersSet* bit to represent step β_1 is completed.

$K(\beta_2)$ - For a scaler node, β_2 first stipulates the maximum *clp* as scaler of each sites, which is achieved simply by traversing all *clp* values of the site. The scaler is then divided by all *clp* of the site. The kernel employs multiple threads to traverse the corresponding *clp* and scale them by scaler in parallel. The scaled *clp* values are stored in global memory for $K(\beta_3)$, as well as the logarithm of scaler of each site.

$K(\beta_3)$ - The *lnscaler* value of each site belonging to the

```
// input <tree node *p>;
// output <scaler pass>;
val shortcut := 0;
val scalerSet := Bit(&p->scalersSet);
val scalerNode := p->scalerNode;
If scalerSet is 1, Then
  shortcut := shortcut bit OR with 1;
EndIf
If scalerNode is YES, Then
  shortcut := shortcut bit OR with 2;
EndIf
//codes used to get lnScaler;
//codes used to get scPOld;
//codes used to get scPNew;
Case shortcut of
  Case 1: shortcut is 1
    scaler_1(lnScaler, scPOld);
  Case 2: shortcut is 2
    scaler_2(lnScaler, scPNew);
  Case 3: shortcut is 3
    scaler_3(lnScaler, scPold, scPNew);
EndCase
```

Figure 5. Pseudo-code that helps decide scaler pass.

phylogeny should be added by the new scales produced in $K(\beta_2)$, the final *lnscaler* need to be added to the likelihood of relevant site of the phylogeny. Similarity, alongside the GPU, CPU process flips the *scalerNode* bit to represent step β_2 is completed.

It can be seen that α_2 have data dependency with α_1, and β_3 has dependency with β_1 and β_2 as well, thus the intermediate results that produced by each kernel stored in global memory firstly by one kernel and read from it again by other kernels is redundant. Besides, as the arithmetic intensity that performed in $K(\alpha_2)$, $K(\beta_1)$ and $K(\beta_3)$ is relatively low, the time consumed on kernel launches and memory access may offset the performance benefit achieved by GPU parallel computing. To reduce the run-time that consumed on kernel launches and redundant data transfers, Ling et al. [11] proposed a tight GPU implementation, namely tgMC^3 method, by merging all these discretely launched GPU kernels into a single kernel. Within the kernel, five corresponding device functions are designed to complete each step, rather than implementing these steps by discrete kernels. This strategy is extremely useful for small datasets with relatively short sequences and few numbers of taxa. However, the scalability of their method is not considerable, as the DOP aforementioned in section 3.1 in the tgMC^3 method is poor. Moreover, it is hard to achieve the best performance by allocating a fixed number of block threads to implement steps with different arithmetic intensity, detailed of this issue will be given in section 3.3.

To draw the experience of Ling's method and also achieve the best DOP, we propose a coarse-grained GPU kernel arrangement strategy by merging α_1 and α_2 in a single kernel, and encapsulating β_1, β_2, and β_3 in another single kernel (see Figure 4(b)). The bit switch operations can be stacked and implemented together by CPU process before and after the launches of these kernels.

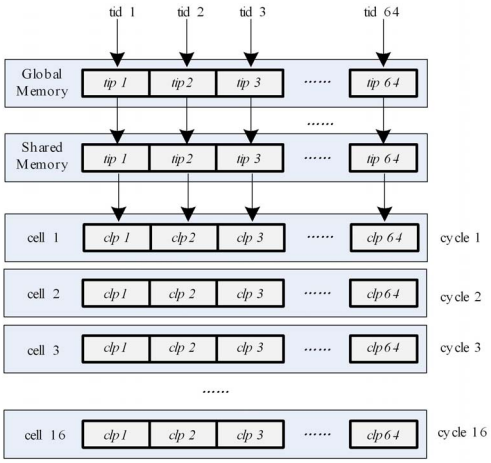

Figure 6. The computation of *clp* is performed recursively by 64 threads.

Since module Φ may not contain β_1, β_2, and β_3, we develop a simple method helping CPU process decide which kernel function is launched to perform the module. As illustrated in Figure 5, *scaler_1*, *scaler_2* and *scaler_3* are corresponding to $K(\beta_1)$, $K(\beta_1\&\beta_2)$, $K(\beta_1\&\beta_2\&\beta_3)$. Previously, five kernels are launched by CPU process to perform module Ψ and Φ. The improve system however requires two GPU kernels to perform these modules, which not only reduced the run-time consumed on data transfer and kernel launches, but also provide a implementation architecture allowing for better scalability.

3.3 Efficient thread arrangement strategy

Under the specified evolutionary parameters, 4 *tip* matrices corresponding to gamma categories are generated to approximate the rate variation among sites. Since the size of each transition matrix is 16 for nucleotide sequences, the overall number of *tip* is 4×16. Given the input sequences of length L and the number of possible state of nucleotide residue 4, the number of *clp* that need to be calculated for each internal node is $4\times4\times L$. Practically, the granularity of *clp* is much larger than *tip*. Hence, the potential parallelizable tasks in two steps are not the same. A critical problem is to define suitable size of thread blocks to fit the number of parallelizable tasks.

In the previous GPU-based implementations, two thread arrangement approaches are introduced. The first approach allocates 64 threads per block, use these threads transfer and calculate the same number of *tip* and *clp* in parallel. The advantage of the approach is no overhead consumed on intra-warp branches. However, it reduces the efficiency of GPU hardware usage and meets relatively larger overhead on data transfers, as the global memory access among warps cannot be well overlapped due to a small number of warps. Another approach is to allocate as many as possible threads, which however adds the cost of branch predictions for GPU threads when transforming *tip*. In addition, the duplicated step by transferring the same *tip* matrices for the calculation of *clp* needs to be performed in both

approaches every time.

To eliminate duplication of effort in data transferring, we developed an efficient thread arrangement strategy by dividing the memory storing *clp* into cells of size equal to the number of *tip*, i.e. 64. The dimension of each thread block is defined as the pattern of (4, 4, 4), where these values stands for the number of model states, gamma rates and terminal states. Whereas GPU computation of *clp* is as follow: Firstly, each block transfers *tip* from global to shared memory. Afterwards, it calculates the *clp* of 16 cells one by one in 16 cycles serially. In each cycle, 64 *clp* are computed by 64 threads in parallel (see Figure 6). The memory used to store *clp* is padded with an appropriate number of dummy symbols if the memory size is not a multiple of 1024 (64×16). The essential idea of the proposed approach for computing Ψ in parallel is to allocate fewer numbers of blocks, but process more *clp* within each block. Assume that there are *M clp* and *N* threads per block, the number of blocks in previous attempts is *M/N*, which becomes to $M/(16\times N)$ under the optimization. The number of repetition is thus reduced to one sixteenth of the previous approaches. The thread arrangement strategy is efficient and significantly eliminates the redundancy of the GPU-based application system. However, the strategy is not suitable for the sequences with relatively short length as GPU resource (stream multiprocessors) are not fully filled with such small number of threads (64) within each block. Once the length of input sequence (the number of unique sites) is shorter than 300, the performance of using more blocks, with each block process single cell achieves slightly better performance. Nevertheless, the proposed strategy achieves better performance and scalability with the growth of the length of sequences.

3.4 Code level optimizations

Coalesced global memory access - During the execution of MrBayes MC^3, the global memory is used to store the intermediate results. The *clp* of descendent nodes should be employed in the computation of their ancestral node. To gain maximum bandwidth and best performance, the intermediate *clp* in global memory should be coalesced access by all threads and saved in shared memory. In module Ψ, 64 terminal states and 1024 intermediate *clp* of descendant nodes are read from global memory by multiple threads in a coalesced manner before they are used in the loop. The computed *clp* in each loop is also able to be written back to global memory in a coalesced manner. Similarly, the strategy is also adopted in module Φ.

Cell block division method - Due to various lengths of sequences are performed in goMC^3 method, and the number of threads in a thread block normally equals to a multiple of warp size, it is common that the number of parallelizable tasks is not a multiple of the number of threads. To avoid memory access leakage, additional branches have to be added in kernels, resulting in a detrimental effect on performance. To design a concise kernel, input sequences are padded with Null Entries (NEs) to have the number of unique sites of a multiple of 64. As a consequence, the size of memory space employed to save *clp*, tree scalers and site likelihood is a multiple of 1024, 64 and 64, respectively. Once the likelihood calculation of a phylogeny is finished, the likelihood of real sites is adopted, while sites with NEs are ignored.

Table 2. The experimental environment of host and device in the proposed method.

GPP (host)	GPU (device)
Operating system: Red Hat Enterprise Linux Server release 6.0 CPU: AMD Phenom(tm) 9850 Quad-Core Processor (4 cores, 2.5GHz) Memory: 8G	Graphic Driver: NVIDIA Driver version 4.2 GPU: NVIDIA Tesla C2075 (14×32 cores, 1.15 GHz) GPU memory: 6GB

4. Experiments

In addition to the multi-core MrBayes MC^3, four publicly available GPU implementations of MrBayes, i.e. oMC^3 (v1.0), nMC^3 (v2.1.1), tgMC^3 (v1.0) and goMC^3 (proposed herein) are used for performance comparative study. Since the aMC^3 [10] is an old version of nMC^3 (v2.1), we will not include the performance of the method. Instead, the state-of-the-art versions of these GPU attempts are tested. Besides, we notice that an open API library, namely BEAGLE [19], is developed for high performance phylogeny. Since the BEAGLE is only supported in MrBayes (v3.2), instead of the version that the aforementioned GPU methods applied on, we will not include the performance of BEAGLE in the experiment. Even though, the performance achieved by BEAGLE on MrBayes (v3.2) is included in supporting files (File S1), which may reflect the advantages of the proposed method from the other side.

4.1 Experimental environment

All experiments are benchmarked on the same node from the GPU cluster maintained at Supercomputing Center, Chinese Academy of Sciences. The details of the experimental environments are listed in Table 2. The gcc version 4.4.4 with option *–O3* is used for compiling MrBayes MC^3, as well as the CPU-side code of oMC^3, nMC^3, tgMC^3 and goMC^3. The GPU-side code of oMC^3, nMC^3, tgMC^3 and goMC^3 are compiled using CUDA Toolkit, version 4.2. The installation path of CUDA SDK should be imported to environment before making the codes, more details can be found in the user manual of goMC^3 (File S2).

4.2 Datasets

Five sets of empirical nucleotide datasets are used in experiments, which can be placed into three categories according to their number of unique sites: short (datasets 1 and 2), medium (dataset 3 and 4) and long sequence length (dataset 5). We briefly list them in Table 3, the details of which can be found in [20, 21]. Besides, 30

Table 3. The list of empirical datasets.

Data set	No.of Taxa	No.of Unique Sites	Description
1	26	353	Trichophora 18S rDNA
2	37	926	Euhemiptera 18S rDNA
3	111	1124	metazoan 18S rDNA
4	234	1697	eukaryotic 18S rDNA
5	288	3177	23 - 28S rDNA from Bacteria, Archaea, and Eukaryota

sets of datasets with simulated sequences generated by Seq-gen v1.3.2 [22] were applied to benchmark the scalability of these implementations. These simulated datasets can be classified into two groups, namely *simu_1* and *simu_2*. Each group has the same number of datasets. Datasets in *simu_1* consist 60 taxa, but a various number of unique site, L, where L {1000, 2000, …, 15000}. Datasets in *simu_2* consist 7500 unique site, but a various number of taxa, N, where N {4, 8, 12, …, 60}.

4.3 Comparative implementation results

It should be noted that the oMC^3 method supports a hybrid computation strategy to improve the overall performance by using additional CPU cores. However, in order to evaluate the performance under the same conditions, all GPU-based methods employ the same number of CPU processes and GPUs for fair comparison. Comparison criteria include the run-time, speedup, and scalability of the resulting implementations.

Run-time. The run-time gives elapsed time for the whole analysis. Table 4 presents the run-time required to analyze the five empirical datasets on the platform described in Table 2, with 1) sequential MrBayes MC^3 using one CPU process, 2) parallel MrBayes MC^3 using four concurrent CPU processes, 3) oMC^3 using one Tesla C2075 card, 4) tgMC^3 using one Tesla C2075 card, 5) nMC^3 using one Tesla C2075 card, 6) goMC^3 using one Tesla C2075 card. The table also includes the peak performance of goMC^3 when using two cards. Table 4 shows the run-time of each method for running 100,000 generations. As a consequence, goMC^3 takes the least run-time in all tests, except for Dataset1 by tgMC^3.

Speedup. We made the following definitions:

- The *X-M speedup* is the number of times faster that X, X {oMC^3, nMC^3, tgMC^3, goMC^3} using one GPU achieves when compared with GPP based MrBayes MC^3 implemented by either one CPU process or four concurrent CPU processes.
- The *G-Y speedup* is the number of times faster that the proposed goMC^3 method using one GPU achieves when compared with another GPU based method Y, Y {oMC^3, nMC^3, tgMC^3}.

Table 5 presents the corresponding speedup factors. For *X-M speedup*, the speedup factors achieved by the four GPU solutions are normalized with respect to the GPP implementation results. As a consequence, the goMC^3 method outperforms the sequential MrBayes MC^3 (1 core) by a factor from 4.7 to 48.1. The corresponding

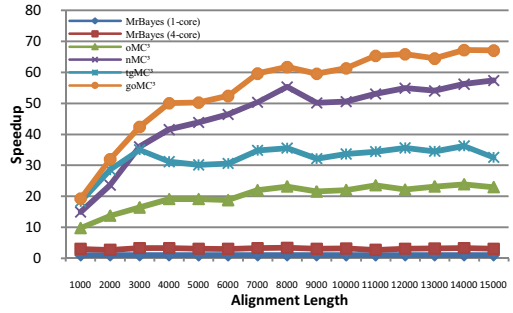

Figure 7. Speedup comparisons between MrBayes (1-core), MrBayes (4-core), oMC^3, nMC^3, tgMC^3 and goMC^3 on a group of datasets, consisting 60 taxa and L unique sites, where $L \in$ {1000, 2000, 3000, …, 15000}.

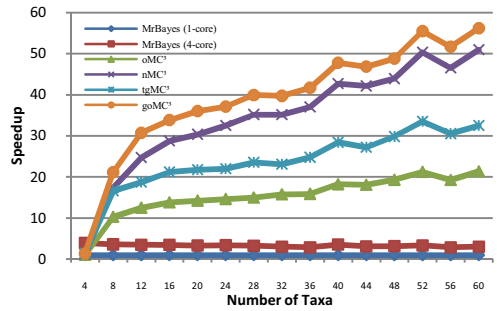

Figure 8. Speedup comparisons between MrBayes (1-core), MrBayes (4-core), oMC^3, nMC^3, tgMC^3 and goMC^3 on a group of datasets, consisting N taxa and 7500 unique sites, where $N \in$ {4, 8, 12, …, 220}.

speedup factors that achieved by oMC^3, nMC^3 and tgMC^3 are 3.2 to 19.2, 3.7 to 39.9, and 4.9 to 38.5, respectively. The performance increases to multi-core MrBayes MC^3 of the four GPU solutions is decreased proportionally. For *G-Y speedup*, experimental results indicate that the goMC^3 method outstrip nMC^3 with speedup factors from 1.21 to 1.46×, oMC^3 from 1.47 to 2.5×, and tgMC^3 from 0.97 to 1.75×, depending on the alignment length. Once again, the tgMC^3 method is very competitive for processing short sequence (dataset1), which is slightly better than the proposed goMC^3 method. For all the rest of tests, goMC^3 always achieves the greatest speedup factor, which highlights the efficiency of the goMC^3 solution. In addition, a peak speedup factor around 77× can be achieved by the

Table 4. Time performance comparison.

Dataset	Alignment length	No.of generation	Run-time (seconds)						
			MrBayes 1 core	MrBayes 4 cores	oMC^3 1 GPU	tgMC^3 1 GPU	nMC^3 1 GPU	goMC^3 1 GPU	goMC^3 2 GPUs
1	1546	100,000	524	154	163	108	143	111	61
2	2238	100,000	1799	544	243	193	179	139	77
3	1506	100,000	4172	1386	502	414	374	257	148
4	1790	100,000	17913	5668	1189	988	773	562	313
5	3386	100,000	38891	13779	2023	1011	975	808	506

Table 5. Speedup comparison of GPU-based solutions to GPP-based MrBayes MC^3 and the performance increases of the proposed goMC^3 method to the other three GPU methods.

Dataset	Speedup (times)										
	O-M		*N-M*		*T-M*		*G-M*		*G-O*	*G-N*	*G-T*
	1-core	4-core	1-core	4-core	1-core	4-core	1-core	4-core			
1	3.2	0.9	3.7	1.1	4.9	1.4	4.7	1.4	1.47	1.28	0.97
2	7.4	2.2	10.1	3.0	9.3	2.8	12.9	3.9	1.74	1.29	1.38
3	8.3	2.8	11.2	3.7	10.1	3.4	16.2	5.4	1.96	1.46	1.61
4	15.1	4.8	23.2	7.3	18.1	5.7	31.9	10.1	2.11	1.37	1.75
5	19.2	6.8	39.9	14.1	38.5	13.6	48.1	17.0	2.50	1.21	1.25

goMC^3 method when scaling it to two Tesla C2075 cards on relatively large dataset with long sequences (dataset 5).

Scalability. The alignment length discussed in this section is the number of unique sites per dataset, as opposed to the complete length of sequences as only unique sites affect the computation complexity of MrBayes MC^3. The speedup factors of other accelerations are normalized by the sequential MrBayes MC^3. Figure 7 illustrates the speedup comparison between these methods when analyzing *simu_1*, the datasets contained in which is composed of a fixed number of taxa, but different alignment length. In each case, we run MrBayes (1-core), MrBayes (4-core), oMC^3, nMC^3, tgMC^3 and goMC^3 for 10,000 generations. In particular, goMC^3 runs roughly 1.13 to 1.36× faster than nMC^3, 1.42 to 2.1× faster than oMC^3 and 1.05 to 2.1× faster than tgMC^3. The speedup achieved by the five acceleration methods on *simu_1* shows that goMC^3 scales the best. Figure 8 illustrates the speedup comparison between these methods when analyzing *simu_2*, the datasets contained in which is composed of a fixed number of alignment length, but different number of taxa. Figure 8 indicates that goMC^3 achieves better scalability than oMC^3 and tgMC^3, but similar with nMC^3. From the view of speed, goMC^3 outperforms nMC^3 with factors from 1.10 to 1.24 in all 15 tests, oMC^3 with factors from 1.30 to 2.68 in all 15 tests, and tgMC^3 with factors from 1.27 to 1.73 in 14 tests, except for the dataset with only 4 taxa. Here again, for uncommonly used small dataset, tgMC^3 achieves slightly better than goMC^3. Nevertheless, the performance that achieved by goMC^3 still outstrips other accelerations in the rest of all tests. The detailed run-time of these tests can be found in supporting information (File S3).

5. Discussion

In order to further exploit the compute capability of GPUs to accelerate MrBayes, we propose goMC^3 approach with three key features: i) the multiple granularity parallel model achieving high degree of parallelism (DOP); ii) a coarse-grained kernel system encapsulating conditional likelihood probability estimation by a smaller number of kernels, thus reducing the run-time consumed on GPU devices memory access and kernel launches; iii) a efficient thread arrangement strategy eliminating the number of redundant operations; In addition, some code level optimizations are applied to fully utilize the GPU hardware and avoid intra-warp branches. As a consequence, goMC^3 can outperforms the sequential MrBayes MC^3 by significant speedup factors both on empirical and simulated datasets and outstrips other GPU-based publicly available implementations.

Figure 7 illustrates the outperformance of goMC^3 compared to other acceleration methods with datasets having the same number of taxa, but various alignment lengths (unique sites). From the speedup point of view, the multi-core based acceleration shows a relatively stable scalability as the alignment length increases, which gives roughly 3× more performance over the sequential version by using 4 concurrent CPU processes. The four GPU based accelerations display a similar trend when analyzing alignment shorter than 3,000. From 1,000 to 3,000 nt, the speedups increase readily from 10 to 16× for oMC^3, 15 to 36× for nMC^3, 18 to 35× for tgMC^3 and 19 to 42× for goMC^3. However, the tgMC^3 method first reaches the plateau of the target platform, as no further performance benefit can be gained over the sequential MrBayes MC^3 when analyzing alignment from 3,000 to 15,000 nt. The oMC^3 method shows similar performance trend with the tgMC^3 method, despite it is slightly better. Instead, both goMC^3 and nMC^3 still gain speedups for extremely long length of sequences.

Figure 8 illustrates the outperformance of goMC^3 compared to other acceleration methods with datasets having the same number of alignment lengths, but various numbers of taxa (except for the dataset having 4 taxa). Obviously, the performance achieved by GPU-based method is negligible when analyzing datasets having extremely small number of taxa (4 taxa) and the performance benefit achieved in each case is relatively smaller than the speedups illustrated in Figure 7. The reason is associated with the number of parallelizable nodes, as which depends on the number of taxa. Precisely, we pick up two datasets from simulated datasets for comparisons. The first comparative dataset is composed of 60 taxa, and each with 5,000 unique sites. The second comparative dataset is composed of 40 taxa, but each with 7,500 unique sites. The speedup factors achieved by four GPU methods for the first comparative dataset are 19 (oMC^3), 30 (tgMC^3), 44 (nMC^3) and 50 (goMC^3), while the factors become to 18 (oMC^3), 28 (tgMC^3), 43 (nMC^3) and 48 (goMC^3) for the second comparative dataset. The comparative results indicate the degree of parallelism is more related to the number of internal nodes that could be potentially parallelized among chains if the number of likelihood probability computation is identical.

We note finally that MrBayes goMC^3 scales extremely well with data sizes both on alignment length and number of taxa. In view of the above, the following conclusion section summarizes the findings of this study.

6. Conclusion

This paper showed the design and implementation of MrBayes MC^3 on off-the-shelf GPUs. The proposed method was compared with a standard sequential GPP implementation and some other publicly available GPU implementations. Comparison criteria included run-time, speedup and scalability. This study revealed that GPUs to be a cost effective and performance efficient platform for the implementation of MrBayes MC^3. The proposed $goMC^3$ algorithm achieved a speedup of up to 67× on single Tesla C2075 graphic card, in comparison to the GPP based sequential MrBayes MC^3. Moreover, $goMC^3$ outperformed the state-of-the-art nMC^3 by a speedup of up to 1.46×, $tgMC^3$ by a speedup of up to 1.75× and oMC^3 by a speedup of up to 2.5×. In addition, $goMC^3$ showed better scalability to the oMC^3 and $tgMC^3$ methods for datasets with increasing alignment length and increasing number of taxa. To conclude, we proposed so far the best solution of implementing MrBayes on GPUs. The solution is composed of a series of steps including a multiple granularity parallelism model, coarse-grained GPU kernel system, efficient thread arrangement and some GPU code level optimizations. These steps improve the degree of parallelism of MrBayes MC^3, reduce the kernel launch complexity, eliminate the number of redundant operations on transition probability, and achieve further performance by harnessing shared memory, global memory coalesced access method, as well as avoiding the branches in GPU kernel. The reported performance improvement of $goMC^3$ method is significant and appears to scale well. In addition, the core calculation of the $goMC^3$ method can be used to benefit the GPU-side code of the oMC^3 method to achieve a more powerful CPU-GPU heterogeneous computation for high performance phylogeny.

Supporting information

File S1. The speedup of Beagle based MrBayes MC^3
File S2. User manual
File S3. Run-time comparisons between the methods
All these files are downloadable from the website in section of availability.

Availability

An open source MrBayes $goMC^3$ (GNU GPL v.2), compilation instructions and the datasets used in the experiments are freely available at: https://github.com/s0897918/MrBayes_goMC3

References

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

[2]. Hastings, W.K. (1970) Monte Carlo Sampling Methods Using Markov Chains and Their Applications. Biometrika 57(1): 97–109.

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

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

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

[6]. 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.

[7]. Van der Wath, R.C. et al. (2008) Bayesian phylogeny on grid. In Bioinformatics Research and Development,13: 404–416.

[8]. 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.

[9]. Zhou, J.F., Liu, X.G., Stones, D.S., Xie, Q., Wang, G. (2011) MrBayes on a Graphics Processing Unit. Bioinformatics, 27(9): 1255-1261.

[10]. Bao, J., Xia, H.J., Zhou, J.F., Liu, X.G., Wang, G. (2013) Efficient Implementation of MrBayes on multi-GPU. Mol Biol Evol, 30(6), 1471–1479.

[11]. Ling, C., Hamada, T., Bai, J.N., Li, X.B., Chester, D., Zheng, W.M., Shi, W.F. (2013) MrBayes $tgMC^3$: A Tight GPU Implementation of MrBayes. PLoS ONE, 8(4), e60667.

[12]. Chai, J., Su, H.Y., Wen, M., Cai, X., Wu, N., Zhang, C.Y. (2013) Resource-efficient utilization of CPU/GPU-based heterogeneous supercomputers for Bayesian phylogenetic inference. J. Supercomput, 66, 364-380.

[13]. Jukes, T. H., Cantor, C. R. (1969) Evolution of protein molecules. In Mammalian protein metabolism (ed. H. N. Munro), Academic Press, pp. 21–123.

[14]. Hasegawa, M., Kishino, H., Yano, T. (1985) Dating the human–ape splitting by a molecular clock of mitochondrial DNA. J. Mol. Evol. 22:160–174.

[15]. Tavaré, S. et al. (2002) Using the fossil record to estimate the age of the last common ancestor of extant primates. Nature 416:726–729.

[16]. Yang, Z. (1993) Maximum likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. Molecular Biology and Evolution 10:1396–1401.

[17]. Yang, Z. (1994) Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. Molecular Evolution 39:306–314.

[18]. Manual of mrbayes: http://mrbayes.sourceforge.net/manual.php

[19]. Ayres, D. et al. (2011) BEAGLE: an application programming interface and high-performance computing library for statistical phylogenetics. Syst Biol. 61(1):170–173.

[20]. Xie, Q., Bu, W., Zheng, L. (2008) 18S rRNA hyper-elongation and the phylogeny of Euhemiptera (Insecta: Hemiptera). Mol Phylogenet Evol 47: 463–471.

[21]. Xie, Q., Wang, Y., Lin, J., Qin, Y., Wang, Y. et al. (2012) Potential key bases of ribosomal RNA to Kingdom-specific spectra of antibiotic susceptibility and the possible Archaeal origin of Eukaryotes. PLoS ONE 7(1): e29468.

[22]. Rambaut, A., 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.