BIOINFORMATICS ORIGINAL PAPER Vol. 25 no. 11 2009, pages 1370–1376
doi:10.1093/bioinformatics/btp244

Phylogenetics

Many-core algorithms for statistical phylogenetics

Marc A. Suchard[1,2,3,*] and Andrew Rambaut[4,*]

[1]Department of Biomathematics, [2]Department of Biostatistics, [3]Department of Human Genetics, University of California, Los Angeles, CA 90095, USA and [4]Institute of Evolutionary Biology, University of Edinburgh, Edinburgh, EH9 3JT, UK

Received on March 16, 2009; revised on April 2, 2009; accepted on April 4, 2009

Advance Access publication April 15, 2009

Associate Editor: Martin Bishop

ABSTRACT

Motivation: Statistical phylogenetics is computationally intensive, resulting in considerable attention meted on techniques for parallelization. Codon-based models allow for independent rates of synonymous and replacement substitutions and have the potential to more adequately model the process of protein-coding sequence evolution with a resulting increase in phylogenetic accuracy. Unfortunately, due to the high number of codon states, computational burden has largely thwarted phylogenetic reconstruction under codon models, particularly at the genomic-scale. Here, we describe novel algorithms and methods for evaluating phylogenies under arbitrary molecular evolutionary models on graphics processing units (GPUs), making use of the large number of processing cores to efficiently parallelize calculations even for large state-size models.

Results: We implement the approach in an existing Bayesian framework and apply the algorithms to estimating the phylogeny of 62 complete mitochondrial genomes of carnivores under a 60-state codon model. We see a near 90-fold speed increase over an optimized CPU-based computation and a >140-fold increase over the currently available implementation, making this the first practical use of codon models for phylogenetic inference over whole mitochondrial or microorganism genomes.

Availability and implementation: Source code provided in BEAGLE: Broad-platform Evolutionary Analysis General Likelihood Evaluator, a cross-platform/processor library for phylogenetic likelihood computation (http://beagle-lib.googlecode.com/). We employ a BEAGLE-implementation using the Bayesian phylogenetics framework BEAST (http://beast.bio.ed.ac.uk/).

Contact: msuchard@ucla.edu; a.rambaut@ed.ac.uk

1 INTRODUCTION

The startling, recent advances in sequencing technology are fueling a concomitant increase in the scale and ambition of phylogenetic analyses. However, this enthusiasm belies a fundamental limitation in statistical phylogenetics: as the number of sequences increases, the size of parameter space—specifically the number of possible phylogenetic histories—explodes. To make matters worse, under any modestly realistic model of sequence evolution, considerable computational effort is required to evaluate each history. Troubling, this effort also skyrockets with the number of sequences and complexity of the sequence characters from nucleotides, through amino acids to codons. Although statistically efficient techniques such as Markov chain Monte Carlo (MCMC) help limit computation effort by concentrating evaluations on only those histories that make a significant contribution to the posterior probability density, improved MCMC methods alone cannot mitigate the non-linear nature of the increase in computational burden. Fortunately, the power of computers available to biologists, famously, has also been growing exponentially with remarkable consistency (Moore, 1998) over a similar time scale to the advances in sequencing technology. One aspect of this is the ubiquitous availability of multi-processor and multi-core computers, inviting novel parallel algorithms to make efficient use of these machines.

The last decade has seen a rapid adoption of parallel computing for molecular phylogenetics (Altekar *et al*., 2004; Feng *et al*., 2003, 2007; Keane *et al*., 2005; Minh *et al*., 2005; Moret *et al*., 2002; Schmidt *et al*., 2002; Stamatakis *et al*., 2005). Concentrating largely on advances for clusters of networked computers, researchers mix and match from a number of parallelization approaches. The first distributes separate histories across multiple computers or CPU cores for independent evaluation (Keane *et al*., 2005; Moret *et al*., 2002; Schmidt *et al*., 2002). The second partitions the sequence data into conditionally independent blocks for distribution across the cluster (Feng *et al*., 2003; Stamatakis *et al*., 2005). The third simultaneously runs multiple MCMC samplers in a synchronized fashion (Altekar *et al*., 2004; Feng *et al*., 2003). Recently using a data partitioning approach, Feng *et al*. (2007) even demonstrate success on a tera-flop cluster, achieving almost linear speedup with the number of nodes employed. However, cluster-based approaches carry with them non-negligible computational over-head in the communication between parallel processes and, critically, linear speedup in the number of CPU processing cores leads to considerable financial costs to purchase hardware or rent super-computer time.

There exists, however, a much less expensive resource available in many desktop computers, the graphics processing unit (GPU), largely unexploited for computational statistics in biology (Charalambous *et al*., 2005; Manavski and Valle, 2008). GPUs are dedicated numerical processors designed for rendering 3D computer graphics. In essence, they consist of hundreds of processor cores on a single chip that can be programmed to apply the same numerical operations simultaneously to each element of large data arrays under a single instruction, multiple data (SIMD) paradigm. Because the same operations, called kernels, function simultaneously, GPUs

*To whom correspondence should be addressed.

can achieve extremely high arithmetic intensity if one can transfer the input data and output results onto and off of the processors quickly. An extension to common programming languages [CUDA: (NVIDIA, 2008)] opens up the GPU to general purpose computing, and the computational power of these units has increased to the stage where they can process data-intensive problems many orders of magnitude faster than conventional CPUs.

Here, we describe novel algorithms to make efficient use of the particular architecture of GPUs for the calculation of the likelihood of molecular sequence data under continuous-time Markov chain (CTMC) models of evolution. Our algorithms build upon Silberstein *et al*. (2008), who demonstrate efficient on-chip memory caching for sum–product calculations on the GPU. Our approach is fundamentally different from previous phylogenetic parallelization, including an exploit of GPUs (Charalambous *et al*., 2005), as we exploit optimized caching, SIMD and the extremely low overhead in spawning GPU threads to distribute individual summations over the unobserved CTMC states within the sum–product algorithm. With this significantly higher degree of parallelization, we demonstrate near 90-fold increases in evaluation speed for models of codon evolution. Codon substitution models (Goldman and Yang, 1994; Muse and Gaut, 1994) decouple the rates of substitution between amino acids and those between nucleotides that do not alter the protein sequence. Such models have the potential to provide a more accurate description of the evolution of protein-coding nucleotide sequences (Shapiro *et al*., 2006) and thus more accurate phylogenetic analyses (Ren *et al*., 2005), but to date have largely only been practically employed to make inferences conditioned on a single phylogenetic history.

2 METHODS

To harness the hundreds of processing cores available on GPUs in phylogenetics, we introduce many-core algorithms to compute the likelihood of n aligned molecular sequences $\mathbf{D}$ given a phylogenetic tree τ with n tips and a CTMC model that characterizes sequence evolution along τ and allows for rate variation along the alignment. To describe these novel algorithms, we first review standard approaches to the data likelihood, $\Pr(\mathbf{D})$, computation (Felsenstein, 1981; Lange, 1997). We then highlight two time-consuming steps in these approaches: (i) computing the probabilities of observing two specific sequences at either end of each branch in τ and (ii) integrating over all possible unobserved sequences at the internal nodes of τ. Finally, we demonstrate how massive parallelization of these steps generates substantial speedup in computing $\Pr(\mathbf{D})$.

2.1 Computing the marginalized data likelihood

The data $\mathbf{D}=(\mathbf{D}_1,\ldots,\mathbf{D}_C)$ comprise C alignment columns, where column data $\mathbf{D}_c=(D_{c1},\ldots,D_{cn})$ for $c=1,\ldots,C$ contain one homologous sequence character from each of the n taxa. Each character exists in one of S possible states that we arbitrarily label $\{1,\ldots,S\}$. Relating these taxa, τ is an acyclic graph grown from n external nodes of degree 1, $n-2$ internal nodes of degree 3 and one root node of degree 2. Connecting these nodes are $2n-2$ edges or branches with their associated lengths $\mathbf{T}=(t_1,\ldots,t_{2n-2})$.

To keep track of these nodes and branches during computation, we require several additional labelings. We label the root and internal nodes with integers $\{1,\ldots,n-1\}$ starting from the root and label the tips of τ arbitrarily with integers $\{1,\ldots,n\}$. We let $\mathcal{I}$ identify the set of internal branches and $\mathcal{E}$ identify the set of terminal branches of τ. For each branch $b\in\mathcal{I}$, we denote the internal node labels of the parent and child of branch b by $\psi(b)$ and $\phi(b)$, respectively. We use the same notation for each terminal branch b except $\psi(b)$ is an internal node index, while $\phi(b)$ is a tip index.

Following standard practice since Felsenstein (1981) we assume that column data $\mathbf{D}_c$ are conditionally independent and identically distributed. Thus, it suffices to compute the likelihood for only each unique $\mathbf{D}_c$ and reweigh appropriately using standard data compression techniques. To ease exposition in this section, we assume all C columns are unique. Let $\mathbf{s}=(\mathbf{s}_1,\ldots,\mathbf{s}_C)$ where $\mathbf{s}_c=(s_{c1},\ldots,s_{c,n-1})$ denote the unobserved internal node sequence states. To incorporate rate variation, we use the very popular discretized models (Yang, 1994) that modulate the CTMC for each column independently through a finite number of rates $r\in\{1,\ldots,R\}$. Then, the joint likelihood of the unobserved internal node data and the observed data at the tips of τ given the rates $\mathbf{r}=(r_1,\ldots,r_C)$ becomes

$$\begin{aligned}\Pr(\mathbf{s},\mathbf{D}|\mathbf{r}) &= \prod_{c=1}^{C}\Pr(\mathbf{s}_c,\mathbf{D}_c|r_c)\\ &= \prod_{c=1}^{C}\left[\pi_{s_{c1}}\prod_{b\in\mathcal{I}}P^{(r_c)}_{s_{c\psi(b)}s_{c\phi(b)}}(t_b)\right.\\ &\qquad\left.\times\prod_{b\in\mathcal{E}}P^{(r_c)}_{s_{c\psi(b)}D_{c\phi(b)}}(t_b)\right],\end{aligned}\tag{1}$$

where $\boldsymbol{\pi}=(\pi_1,\ldots,\pi_S)$ are the prior probabilities of the unobserved character in each column at the root node and $P^{(r)}_{sj}(t)$ for $s,j\in\{1,\ldots,S\}$ are the finite-time transition probabilities of character j existing at the end of a branch with length t given character s at the start under rate r. These probabilities derive from the rate-modulated CTMC and we discuss their calculation in a later section. Often, $\boldsymbol{\pi}$ also derives from the CTMC.

Unpacking Equation (1), if we guess the internal node states and rate for each column, the likelihood reduces simply to the product of character transition probabilities over all of the branches in τ and across all columns. However, $\mathbf{s}$ and $\mathbf{r}$ are never observed and, hence, we sit with a further time-consuming integration. Letting $\Pr(r)$ represent the prior probability mass function over rates, we recover the observed data likelihood by summing over all possible internal node states and column rates,

$$\begin{aligned}\Pr(\mathbf{D}) = \prod_{c=1}^{C}\left[\sum_{r=1}^{R}\left(\sum_{s_1=1}^{S}\cdots\sum_{s_{n-1}=1}^{S}\pi_{cs_1}\prod_{b\in\mathcal{I}}P^{(r)}_{s_{c\psi(b)}s_{c\phi(b)}}(t_b)\right.\right.\\ \left.\left.\times\prod_{b\in\mathcal{E}}P^{(r)}_{s_{c\psi(b)}D_{c\phi(b)}}(t_b)\right)\Pr(r)\right].\end{aligned}\tag{2}$$

2.1.1 Traditional computation Judiciously distributing the sum over $\mathbf{s}$ within the product over branches in Equation (2) reduces its computational complexity from exponential to polynomial in n and forms a sum–product algorithm, also known as Felsenstein's Peeling algorithm in phylogenetics (Felsenstein, 1981). This approach invites a post-order traversal of the internal nodes in τ and the computation of a recursive function at each node that depends only on its immediate children; to the best of our knowledge, all likelihood-based phylogenetic software exploits this recursion.

Let $\mathbf{F}_u=\{F_{urcs}\}$ be an $R\times C\times S$ matrix of forward, often called partial or fractional, likelihoods at node u. Element F_{us} is the probability of the observed data at only the tips that descend from node u, given that the state of u is s. If u is a tip, then we initialize partial likelihoods via equation $F_{us}=1_{\{s=D_u\}}$. In case of missing or ambiguous data, D_u denotes the subset of possible character states, and forward likelihoods are set to $F_{us}=1_{\{s\in D_u\}}$. During post-order, or upward, traversal of τ, we compute forward likelihoods for each internal node u using the recursion

$$\begin{aligned}F_{urcs} = &\left[\sum_{j=1}^{S}F_{\phi(b_1)rcj}\times P^{(r)}_{sj}(t_{b_1})\right]\\ &\times\left[\sum_{j=1}^{S}F_{\phi(b_2)rcj}\times P^{(r)}_{sj}(t_{b_2})\right],\end{aligned}\tag{3}$$

Algorithm 1 GPU-based parallel computation of partial-likelihoods F_{urcs} through peeling

1: Define COLUMN_BLOCK_SIZE (CBS) ← number of columns processed per thread block
2: Define STATE_BLOCK_SIZE (SBS) ← number of states processed per inner-loop
3: **for all** thread blocks (rate class $r=1,\ldots,R$ and column-block $=1,\ldots,\lceil C/\mathsf{CBS}\rceil$) in parallel **do**
4: Using each thread $s=1,\ldots,S$ and $c=1,\ldots,\mathsf{CBS}$ in block, prefetch child partial likelihoods $F_{\phi(b_1)rcs}$ and $F_{\phi(b_2)rcs}$ for CBS columns (reused by all threads in block)
5: Initialize $F^{(1)}_{urcs} \leftarrow 0$ and $F^{(2)}_{urcs} \leftarrow 0$
6: **for** $j=1$ to S in SBS increments **do**
7: Pre-fetch transition probabilities $P^{(r)}_{sj}(t_{b_1})$ and $P^{(r)}_{sj}(t_{b_2})$ for SBS states (reused by all threads in block)
8: $F^{(1)}_{urcs} \mathrel{+}= F_{\phi(b_1)rcs} \times P^{(r)}_{sj}(t_{b_1})$
9: $F^{(2)}_{urcs} \mathrel{+}= F_{\phi(b_2)rcs} \times P^{(r)}_{sj}(t_{b_2})$
10: **end for**
11: **end for**
12: Return $F^{(1)}_{urcs} \times F^{(2)}_{urcs}$

where b_1 and b_2 are indices of the branches descending from node u and $\phi(b_1)$ and $\phi(b_2)$ are the corresponding children of u.

Given the final recursive computations F_{1rcs_1} at the root, we recover the data likelihood through a final summation over the root state s_1 and column rate r across all columns,

$$\Pr(\mathbf{D}) = \prod_{c=1}^{C} \left[\sum_{r=1}^{R} \left(\sum_{s_1=1}^{S} \pi_{s_1} \times F_{1rcs_1} \right) \Pr(r) \right]. \tag{4}$$

2.1.2 Many-core computation Computing Equation (3) for all (r,c,s) is generally $O(RCS^2)$ but is at best order $O(RCS)$ when both children are tips and have unambiguous data. In either case, this high computational cost invites parallelization. For each node u in the recursion, we propose distributing these computations across many-core processors such that each (r,c,s)-entry executes in its own short-lived thread. This fine scale of parallelization differs substantially from previous approaches that partition the columns into conditionally independent blocks and distribute the blocks across separate processing cores and is possible because GPUs share common memory and sport negligible costs to spawn and destroy threads, potentiating significant speedup improvement.

Algorithm 1 outlines our implementation of Equation (3). For each (r,c,s)-thread, the algorithm dedicates only a small portion of its code to actually computing F_{urcs}. Most of the work involves efficiently reading and caching the large vectors of child forward likelihoods $\{F_{\phi(b_1)rcj}\}$ and $\{F_{\phi(b_2)rcj}\}$ for $j=1,\ldots,S$ and even larger finite-time transition probability matrices $\{P^{(r)}_{sj}(t_{b_1})\}$ and $\{P^{(r)}_{sj}(t_{b_2})\}$ for $s,j=1,\ldots,S$ to maximize computational throughput.

To maximize data throughput with global memory, the GPU hardware combines, or 'coalesces', memory read/writes of 16 consecutive threads into a single wide memory transaction. If one cannot coalesce global memory access, then separate memory transactions occur for each thread, resulting in high latency. Thus, our algorithm attempts to read/write only multiples-of-16 values at a time. One approach we take embeds models in which S is not a multiple of 16 into a larger space by zero-padding extra entries in the forward likelihoods and transition probabilities. For example, codon models have a number of stop-codons (depending on the exact genetic code being employed), which are not considered valid states within the CTMC. This marginally reduces $S<64$; however, S is no longer a multiple of 16, so we treat the stop-codons as zero-probability states, yielding the full $S=64$. For nucleotide models, we simply process four (r,c,s)-entries, instead of one, in a single thread.

On-board the GPU processing units themselves sits up to 16 KB of memory shared between all threads grouped into a thread-block with a maximum of 512 threads per block. Shared memory performance is between 100- to 150-fold faster than even 'coalesced' global memory transactions. However, 16 KB is very small, about 5400 single-precision values. To optimize performance, we attempt to cooperatively prefetch the largest possible chunks of data sitting in global memory using coalesced transactions and cache these values in shared memory, in an order that maximizes their re-use across the threads in a block. Efficient caching requires carefully partitioning the $R \times C \times S$ threads into blocks.

Two considerations direct our thread-block construction. First, within a rate class and column, F_{urcs} for all states s depend on the same S forward likelihoods for both child nodes that we wish to load from global memory once. Second, within a rate class, F_{urcs} for each column c depends on the same finite-time transition probabilities. Consequentially, we construct $R \times \lceil C/\mathsf{CBS}\rceil$ thread-blocks, where $\lceil\cdot\rceil$ is the ceiling function and column-block size (CBS) is a design constant, controlling the number of columns processed in a block. Each block shares $S \times \mathsf{CBS}$ threads that correspond to all states for CBS columns. Processing multiple columns allows for the reuse of cached finite-time transition probabilities. To this end, we choose CBS as large as possible such that $S \times \mathsf{CBS} \leq 512$. For codon models, $\mathsf{CBS}=8$.

The final complication of Algorithm 1 arises when all S^2 transition probabilities do not fit in shared memory. Instead, the threads cooperatively prefetch columns of the matrix in peeling-block size (PBS) length chunks. We choose PBS as large as possible without overflowing shared memory; for codon models in single precision, $\mathsf{PBS}=8$. To enable coalesced global memory reads of the matrix columns, we exploit a column-wise flattened representation of the finite-time transition matrices; this differs from the standard row-wise representation in modern computing.

2.2 Computing the finite-time transition probabilities

Integral to the data likelihood are the finite-time transition probabilities $P^{(r)}_{sj}(t)$ that characterize how state s mutates to state j along a branch of length t at rate r. Common practice in likelihood-based phylogenetics models the evolution of molecular characters as an irreducible, reversible S-state CTMC with infinitesimal generator $\mathbf{\Lambda} = \mathbf{\Lambda}(\theta)$. Unknown generator parameters θ govern the behavior of the chain. For nucleotide characters, popular parameterizations include the Hasegawa *et al*. (1985, HKY85) and Lanave *et al*. (1984, GTR) models. Due to their computational complexity, codon models remain less explored. Goldman and Yang (1994) and Yang *et al*. (2000, M_0) introduce a $S=61$-state model with parameters $\theta = (\kappa, \omega, \pi)$, where κ measures the transition:transversion rate ratio, ω controls the relative rate of non-synonymous to synonymous substitutions and π models the stationary distribution of characters and is often fixed to empirical estimates.

For many commonly used CTMCs and notably for all codon models, closed-form expression for the finite-time transition probabilities given θ do not exist and researchers exploit numerical eigendecomposition of $\mathbf{\Lambda}$ to recover the probabilities through

$$\begin{aligned}\mathbf{P}^{(r)}(t) &= \exp(\mu_r t \mathbf{\Lambda}) = \mathbf{E} \times \mathrm{diag}(e^{\mu_r t \lambda_1}, \ldots, e^{\mu_r t \lambda_S}) \times \mathbf{E}^{-1} \\ &= \mathbf{E}\mathbf{D}_{rt}\mathbf{E}^{-1}, \end{aligned} \tag{5}$$

where μ_r is the rate multiplier for rate class r and $\mathbf{E}$ is the matrix of eigenvectors of $\mathbf{\Lambda}$ with corresponding eigenvalues $\lambda_1, \ldots, \lambda_S$. These eigenvalues and $\mathbf{E}$ are implicit functions of θ. Generally, one thinks of matrix diagonalization as a rate-limiting step (Schadt *et al*., 1998) in phylogenetic reconstruction, as diagonalization proceeds at $O(S^3)$; however, a typical MCMC sampler attempts to update branch lengths and rate-multipliers several orders of magnitude more often than the sampler updates θ. Consequentially, computation effort in recomputing Equation (5) for each rate class r and branch length t in τ far outweighs diagonalization of $\mathbf{\Lambda}$.

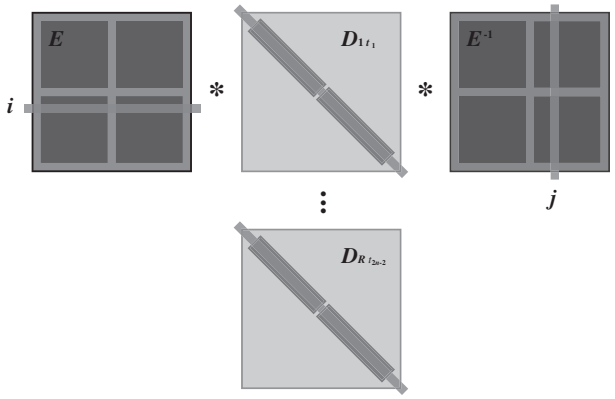

Fig. 1. Parallel thread block design for computing the finite-time transition probabilities $\mathbf{P}^{(r)}(t)=\mathbf{E}\mathbf{D}_{rt}\mathbf{E}^{-1}$ for all R rate classes along all $2n-2$ branches simultaneously. Example assumes that two prefetch blocks (shaded) span the matrices.

2.2.1 Many-core computation While time-consuming diagonalization occurs infrequently, constructing finite-time transition probabilities from a diagonalization is a major bottleneck and parallelization affords advantages in these calculations as well. Specifically, updating infinitesimal parameters θ and one rediagonalization leads to recalculating all probabilities for each rate class r along all $2n-2$ branches in τ. Also, updating an internal node height in τ leads to recalculating the transition probabilities for all R for the two or three branches incident to the affected internal node.

Each of these updates starts with same eigenvector matrices and multiplies these against differently scaled and exponentiated eigenvalues. After precalculating the scaling factors, we perform these matrix multiplications in parallel. Our algorithm is a natural extension to usual GPU-based matrix multiplication kernels for arbitrary matrices $\mathbf{A}$ and $\mathbf{B}$, see e.g. Choi *et al*. (1994) and Lee *et al*. (1997). Since $\mathbf{A}$ and $\mathbf{B}$ are often too large to fetch completely onto the GPU shared memory, these kernels take a block strategy. Each block of threads performs a loop. In each loop step, the threads form the product $\mathbf{A}_{\text{sub}}\mathbf{B}_{\text{sub}}$ of smaller submatrices that do fit within the shared memory of the block. The threads cooperatively prefetch all entries in $\mathbf{A}_{\text{sub}}$ and $\mathbf{B}_{\text{sub}}$. Then, each thread computes one entry of the product; this depends on many of the prefetched values, and so the kernel maximizes shared memory reuse. The threads tabulate results as the loop moves the submatrices across $\mathbf{A}$ and down $\mathbf{B}$. We modify this algorithm in two ways. First, the threads within a block prefetch the appropriate submatrix diagonal entries of $\mathbf{D}_{rt}$ and form the three-matrix product; and second, we greatly extend the number of blocks to perform evaluation for all rate classes and branch lengths simultaneously. Figure 1 depicts this blocking strategy when multiplying $\mathbf{E}\mathbf{D}_{rt}\mathbf{E}^{-1}$. The figure assumes two blocks span S; in general, we set the block dimension to 16 to ensure coalesced global memory transactions. Finally, greatly speeding memory retrieval in the peeling algorithm, we store each resultant matrix $\mathbf{P}^{(r)}(t)$ in a column-wise, transposed representation in global memory.

2.3 Precision

Graphics rendering normally requires only 32-bit (single) precision floating point numbers and thus most GPUs compute at single precision. While the latest generation of GPUs can operate on 64-bit (double) precision numbers, the precision boost comes with a performance cost because the GPU contains far fewer double-precision units. Further, even at double precision, rounding error can still occur while propagating the partial likelihoods up a large tree. In general, we opt for single-precision floats and implement a rescaling procedure when calculating the partial likelihoods to help avoid roundoff. Our approach follows the suggestion of Yang (2000). When an under- or over-flow occurs while computing the likelihood, we record the largest partial likelihood $M_{uc}=\max_{r,s}F_{urcs}$ for each site c. As we peel up the internal nodes, we replace F_{urcs} with F_{urcs}/M_{uc} to help avoid roundoff. We recover the final likelihood for each column at the root through rescaling the resulting calculations by $\prod_u M_{uc}$. In contrast to Yang (2000), we compute M_{uc} and rescale the partial likelihoods in parallel for each site. When S is a power of 2, we further improve performance by performing the maximum-element search over s through a parallel-reduction algorithm. Until another under- or over-flow occurs, we leave M_{uc} fixed.

2.4 Harnessing multiple GPUs

We provide support for multiple GPUs through a simple load balancing scheme in which we divide the data columns amongst approximately equal partitions and distribute one partition to each GPU. This scheme replicates considerable work in calculating the same finite-time transition probabilities on each GPU but the alternative, computing these values once on a master GPU and then distributing, could be much slower as it requires transferring large blocks of data from card to card; this carries extremely high latency. Our approach also facilitates the distribution of different genomic loci to each GPU whilst allowing each to have different models of evolution (for example, combining mitochondrial and nuclear loci requires different genetic codes and thus different transition probability matrices).

3 EXAMPLE

To illustrate the performance gains that GPUs afford in statistical phylogenetics, we explore the evolutionary relationship of mitochondrial genomes from 62 extant carnivores and a pangolin outgroup, compiled from GENBANK. This genomic sequence alignment contains 10 860 nt columns that code for 12 mitochondrial proteins and when translated into a 60-state vertebrate mitochondrial codon model, yields an impressive 3620 columns, of which 3600 are unique. Conducting a phylogenetic analysis on such an extensive dataset using a codon substitution model would previously be considered computationally foolhardy.

Figure 2 displays the tree for these carnivores that we infer from a Bayesian analysis under the M_0 codon model (Goldman and Yang, 1994) with a 4-class discrete-Γ model for rate variation (Yang, 1994) and relaxed molecular clock (Drummond *et al*., 2006). This analysis uses GPU-enabled calculations through MCMC to draw 25 million random samples from the joint posterior of both an unknown phylogeny and unknown parameters characterizing the substitution model. The resulting phylogeny is compatible with the current understanding of the familial relationships of Carnivora and importantly helps resolve relationships within the Arctoidea. This infraorder within the caniform (dog-like) carnivores comprises the Ursidae (bears), Musteloidea (weasles, raccoons, skunks and red panda) and the Pinnipedia (seals and walruses) and the branching order has been debated. Delisle and Strobeck (2005) entertain a nucleotide-based analysis of a smaller dataset of the same 12 genes and show marginal support for either the grouping of Pinnipedia and Ursidae or Musteloidea and Uridae (depending on whether a Bayesian or maximum likelihood approach, respectively, is employed). When run under a conventional GTR nucleotide model, our alignment also yields the Pinnipedia and Ursidae grouping but with less than emphatic support (posterior probability 0.79). However, with the codon model, the same data proffer significant support (posterior probability 0.97) for the Pinnipedia and Musteloidea grouping, a relationship also found for nuclear markers (Flynn *et al*., 2000; Yu *et al*., 2004)

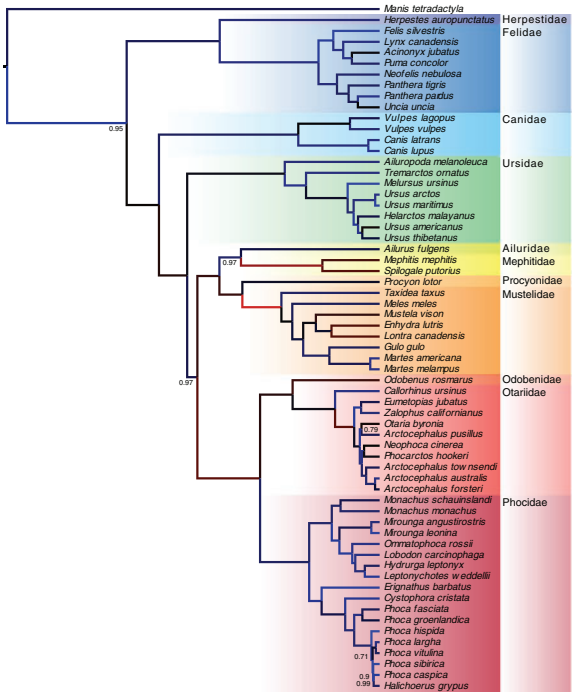

Fig. 2. Reconstructed codon-based majority clade consensus tree of 62 carnivore mitochondrial protein-coding sequences with the long-tailed pangolin (*Manis tetradactyla*) as an outgroup. We label clades with posterior probabilities except where they approach 1 and color branches to reflect relative rates of molecular evolution (red/faster, blue/slower) under a relaxed clock.

Table 1. Codon substitution model parameter estimates for 62 extant carnivores and the pangolin outgroup

	Posterior mean	95% Bayesian credible interval
Tree height	4.2	(2.5–5.5)
Transition:transversion ratio κ	12.1	(11.7–12.4)
dN/dS ratio ω	0.0274	(0.0265–0.0284)
Rate variation dispersion α	1.55	(1.48–1.62)
Relaxed clock deviation σ	0.28	(0.16–0.74)

and 'supertree' approaches that attempt to synthesize available phylogenetic knowledge (Bininda-Emonds *et al.*, 1999).

Table 1 reports the marginal posterior estimates for the tree height in expected substitutions, the transition:transversion rate ratio κ, the non-synonymous:synonymous rate ratio ω, rate variation dispersion α and relaxed clock deviation σ. These estimates demonstrate the high overall rate of synonymous substitution, suggesting that these changes may be saturating faster than amino acid replacement substitutions, an effect not adequately modeled at the nucleotide level. The codon model employed here does allow different rates for synonymous and replacement substitutions and this helps to explain the congruence of the resulting tree with that of nuclear genes that evolve at a slower rate.

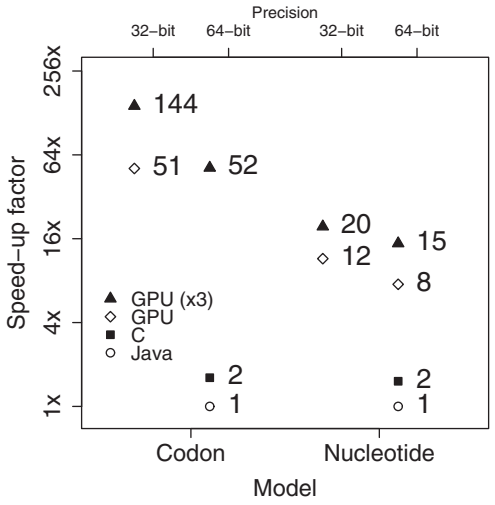

Fig. 3. Speed comparison of GPU, Java and C BEAST implementations. Speedup factors are on the log-scale.

We perform this codon-based analysis on a standard desktop PC sporting a 3.2 GHz Intel Core 2 Extreme (QX9770) CPU and 8 GB of 1.6 GHz DDR3 RAM. This CPU is the fastest available at the end of 2008 and, together with the high-end RAM, maximizes the speed at which the Java and C implementations run to provide a benchmark comparison. We equipped the desktop PC with three NVIDIA GTX280 cards each of which carries a single GPU with 240 processing cores running at 1.3 GHz and 1 GB of RAM. Exploiting all three GPUs, our analysis only requires 64 h to run.

Figure 3 compares runtime speeds using all three GPUs, a single GPU and Java and C implementations. For codon models, the C version is not available in the current BEAST release but our C implementation provides only a 1.6-fold speed increase over Java. To make these comparisons in reasonable time, we run short MCMC chains of 10 000 steps. Between the three GPUs and single GPU for the codon model, we observe a slightly <3-fold improvement. This factor compares well with the theoretical maximum, identifying that the replicated work in recomputing the finite-time transition probabilities is not a burden when fitting models with large state spaces. Between the three GPUs and the current BEAST implementation, we observe an over 140-fold improvement (a 90-fold improvement over the C implementation). One major difference between the default computations on the GPU and the CPU implementations is the precision, as modern CPUs provide double-precision at full performance. Fortunately, the GTX280 GPU also contains a very limited number (30) of double precision floating point units and it is possible to exploit double precision on the GPU, but at a performance cost. For the three GPU configuration, this cost is also ∼3-fold. To compute this data example in double precision requires more than the 1 GB of RAM available on a single card so we are unable to replicate this experiment using a single GPU. This is a limitation of our particular hardware; cards with 4 GB of RAM are now available. However, for this dataset, the performance hit comes with no scientific gain; posterior estimates at both double- and single precision remain unchanged.

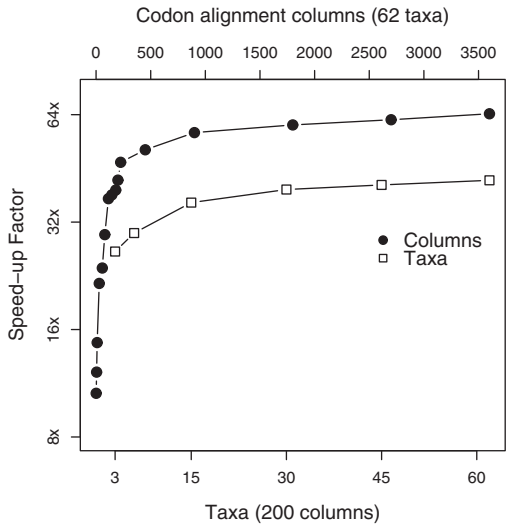

Fig. 4. GPU performance scaling by number of unique alignment columns *C* and of taxa *N*. Speedup factors are on the log-scale.

One important scaling dimension of the parallelization is model state-space size *S*. We reanalyze the carnivore alignment at the nucleotide level using the GTR model for nucleotide substitution. For nucleotides, $S=4$ and we expect the speedup that the GPUs afford to become much more modest. However, we still observe ∼20-fold (three GPUs) and 12-fold (single GPU) performance improvements at single-precision, and single- and double-precision MCMC chains return the same estimates. These substantial increases definitely warrant using the GPUs even for small state spaces.

Two other scaling dimensions of practical concern are the numbers of alignment columns *C* and of taxa *N*. For very small *C*, the overhead in transferring data onto and off of the GPU may outweigh the computational benefit; performance should then rapidly improve to a point where the parallel resources on the GPU become fully occupied. After this saturation point, we suspect only moderate performance increase as *C* grows. The former cross-over point is critical in choosing whether or not to attempt a GPU analysis and the latter provides guidance in splitting datasets across multiple GPUs. On the other hand, changing *N* should have a much less pronounced effect with one additional taxon introducing one internal nodes and two branches to the phylogeny. Figure 4 reports how performance scales when analyzing 1–3600 unique codon alignment columns for all 62 carnivores and 200 columns for 3–62 taxa. Interestingly, even for a single codon column, the GPU implementation runs over 10-fold faster due to the performance gains in our parallel calculation of the finite-time transition probabilities. For this codon model, the saturation point occurs around 500 unique columns; therefore, optimal performance gains arise from distributing several hundred columns to each GPU in a multi-GPU system.

4 DISCUSSION

The many-core algorithms we present in this article capably provide several orders of magnitude speedup in computing phylogenetic likelihoods. Given our algorithm design, performance is most impressive for larger state-space CTMCs, such as codon models. However, even nucleotide models demonstrate marked speed improvements, and our algorithms remain beneficial for small state spaces as well.

Codon models are particularly attractive as they can model the process of natural selection acting on different parts of the gene but are currently too prohibitively slow for use in inferring evolutionary relationships for non-trivial alignments. Through GPU computing, we are now able to infer the relationship among 62 carnivores using the M_0 codon model applied to the entire protein-coding mtDNA genome. For this example, our algorithms compute nearly 150-fold faster than the currently available methods in the BEAST sampler. Using extrapolation to put this number in perspective, the current BEAST implementation would require 1.1 uninterrupted years to complete a 25 million MCMC sample run on our desktop PC, compared with the 64 h we endured.

For researchers looking to improve computational speed, one option is simply to purchase more computers of the same performance and link them into a cluster. At best this provides a linear speedup in speed proportional to the cost of each additional system. Even ignoring the costs of maintenance, space or air conditioning for such an increasingly large system, we suggest that upgrading the existing system with the currently available GPU boards could provide a similar performance at a fraction of the price. However, if budget is of limited concern, it is obviously possible to combine these approaches and equip cluster-computer nodes with GPU processing as well.

While our specific implementation of these algorithms is within a Bayesian framework, maximum likelihood inference requires the same calculations and a large number of software packages could benefit from the utilization of GPU computation. Indeed, we envisage the possibility and desirability of producing a general-purpose library and application programming interface (API) that abstracts the exact hardware implementation from the calling software package. In this way, improvements to these algorithms could be made or they could be implemented on other many-core hardware architecture, and all the supported packages would benefit. To this end, we have started an open source library, BEAGLE: Broad-platform Evolutionary Analysis General Likelihood Evaluator, that provides both an API and implementations in Java, C and for CUDA-based GPUs.

ACKNOWLEDGEMENTS

We thank Alexei Drummond for his impassioned support of the open source programming and critical insight on enabling GPU support in BEAST and Mark Silberstein for early discussions on implementing sum–product algorithms on GPUs. We thank John Welch for compiling the carnivore mitochondrial dataset.

Funding: Research gift from Microsoft Corporation, National Institutes of Health (R01 GM086887 to M.S.); The Royal Society (to A.R.)

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.

Bininda-Emonds,O. *et al*. (1999) Building large trees by combining phylogenetic information: a complete phylogeny of the extant Carnivora (Mammalia). *Biol. Rev.*, **74**, 143–175.

Charalambous,M. *et al*. (2005) Initial experiences porting a bioinformatics application to a graphics processor. In Bozanis,P. Houstis,E.N. (eds), *Proceedings of 10th Panhellenic Conference on Informatics (PCI2005)*, Vol. 3746 of Lecture Notes in Computer Science, Springer, New York, pp. 415–425.

Choi,J. *et al*. (1994) PUMMA: parallel universal matrix multiplication algorithms on distributed memory concurrent computers. *Concurrency Pract. Exp.*, **6**, 543–570.

Delisle,I. and Strobeck,C. (2005) A phylogeny of the Caniformia (order Carnivora) based on 12 complete protein-coding mitochondrial genes. *Mol. Phylogenet. Evol.*, **37**, 192–201.

Drummond,A. *et al*. (2006) Relaxed phylogenetics and dating with confidence. *PLoS Biol.*, **4**, e88.

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 Distr. Comput.*, **63**, 707–718.

Feng,X. *et al*. (2007) Building the Tree of Life on terascale systems. In *Parallel and Distributed Processing Symposium* (IPDPS 2007). Washington, DC.

Flynn,J. *et al*. (2000) Whence the red panda? *Mol. Phylogenet. Evol.*, **17**, 190–199.

Goldman,N. and Yang,Z. (1994) A codon-based model of nucleotide substitution for protein-coding DNA sequences. *Mol. Biol. Evol.*, **11**, 725–736.

Hasegawa,M. *et al*. (1985) Dating the humanape splitting by a molecular clock of mitochondrial DNA. *J. Mol. Evol.*, **22**, 160–174.

Keane,T. *et al*. (2005) DPRml: distributed phylogeney reconstruction by maximum likelihood. *Bioinformatics*, **21**, 969–974.

Lanave,C. *et al*. (1984) A new method for calculating evolutionary substitution rates. *J. Mol. Evol.*, **20**, 86–93.

Lange,K. (1997) *Mathematical and Statistical Methods for Genetic Analysis*. Springer, New York.

Lee,H.-J. *et al*. (1997) Generalized cannon's algorithm for parallel matrix multiplication. In *ICS '97: Proceedings of the 11th International Conference on Supercomputing*. ACM, New York, NY, pp. 44–51.

Manavski,S.A. and Valle,G. (2008) CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics*, **9**, S10.

Minh,B. *et al*. (2005) pIQPNNI: parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, **21**, 3794–3796.

Moore,G. (1998) Cramming more components onto integrated circuits. *Proc. IEEE*, **86**, 82–85.

Moret,B. *et al*. (2002) High-performance algorithm engineering for computational phylogenetics. *J. Supercomput.*, **22**, 99–111.

Muse,S. and Gaut,B. (1994) A likelihood approach for comparing synonymous and nonsynonymous nucleotide substitution rates, with applications to the chloroplast genome. *Mol. Biol. Evol.*, **11**, 715–725.

NVIDIA (2008) NVIDIA CUDA Compute Unified Device Architecture: Programming Guide Version 2.0. Available at http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf (last accessed date March 16, 2009)

Ren,F. *et al*. (2005) An empirical examination of the utility of codon-substitution models in phylogeny reconstruction. *Syst. Biol.*, **54**, 808–818.

Schadt,E. *et al*. (1998) Computational advances in maximum likelihood methods for molecular phylogeny. *Genome Res.*, **8**, 222–233.

Schmidt,H. *et al*. (2002) TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, **18**, 502–504.

Shapiro,B. *et al*. (2006) Choosing appropriate substitution models for the phylogenetic anaylsis of protein-coding sequences. *Mol. Biol. Evol.*, **23**, 7–9.

Silberstein,M. *et al*. (2008) Efficient computation of sum-products on GPUs through software-managed cache. In Zhou,P. (ed.) *Proceedings of the 22nd Annual International Conference on Supercomputing*. ACM, New York, pp. 309–318.

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

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

Yang,Z. (2000) Maximum likelihood estimation on large phylogeneis and analysis of adaptive evolution in human influenza virus A. *J. Mol. Evol.*, **51**, 423–432.

Yang,Z. *et al*. (2000) Codon-substitution models for heterogeneous selection pressure at amino acid sites. *Genetics*, **155**, 431–449.

Yu,L. *et al*. (2004) Phylogenetic relationships within mammalian order Carnivore indicated by sequences of two nuclear DNA genes. *Mol. Phylogenet. Evol.*, **33**, 694–705.