ABSTRACT

Title of dissertation: RESEARCH AND APPLICATION OF PARALLEL COMPUTING ALGORITHMS FOR STATISTICAL PHYLOGENETIC INFERENCE

Daniel L. Ayres, Doctor of Philosophy, 2017

Dissertation directed by: Professor Michael P. Cummings
Department of Biology

Estimating the evolutionary history of organisms, phylogenetic inference, is a critical step in many analyses involving biological sequence data such as DNA. The likelihood calculations at the heart of the most effective methods for statistical phylogenetic analyses are extremely computationally intensive, and hence these analyses become a bottleneck in many studies. Recent progress in computer hardware, specifically the increase in pervasiveness of highly parallel, many-core processors has created opportunities for new approaches to computationally intensive methods, such as those in phylogenetic inference.

We have developed an open source library, BEAGLE, which uses parallel computing methods to greatly accelerate statistical phylogenetic inference, for both maximum likelihood and Bayesian approaches. BEAGLE defines a uniform application programming interface and includes a collection of efficient implementations that use NVIDIA CUDA, OpenCL, and C++ threading frameworks for evaluating likelihoods under a wide variety of evolutionary models, on GPUs as well as on

multi-core CPUs. BEAGLE employs a number of different parallelization techniques for phylogenetic inference, at different granularity levels and for distinct processor architectures. On CUDA and OpenCL devices, the library enables concurrent computation of site likelihoods, data subsets, and independent subtrees. The general design features of the library also provide a model for software development using parallel computing frameworks that is applicable to other domains.

BEAGLE has been integrated with some of the leading programs in the field, such as MrBayes and BEAST, and is used in a diverse range of evolutionary studies, including those of disease causing viruses. The library can provide significant performance gains, with the exact increase in performance depending on the specific properties of the data set, evolutionary model, and hardware. In general, nucleotide analyses are accelerated on the order of 10-fold and codon analyses on the order of 100-fold.

RESEARCH AND APPLICATION OF PARALLEL COMPUTING ALGORITHMS FOR STATISTICAL PHYLOGENETIC INFERENCE

by

Daniel L. Ayres

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2017

Acknowledgments

The work underlying this thesis was carried out at the Center for Bioinformatics and Computational Biology at the University of Maryland, College Park. Financial support from the National Science Foundation is gratefully acknowledged. In addition, NVIDIA Corporation, Advanced Micro Devices, Inc., and Intel Corporation are acknowledged for providing computer resources and technical support.

I am deeply grateful to my supervisor, Professor Michael Cummings, for giving me the opportunity to carry out my doctoral studies at the Laboratory of Molecular Evolution. I thank him for his endless support, scientific guidance, and constructive comments during the work of this thesis.

This work would not have existed without excellent collaboration with other research groups. I am deeply grateful to all my co-authors: Marc Suchard from the University of California, Los Angeles, Andrew Rambaut from the University of Edinburgh, Fredrik Ronquist from the Swedish Museum of Natural History, Derrick Zwickl from the University of Arizona, Aaron Darling from the University of Technology Sydney, John Huelsenbeck from the University of California, Berkeley, Paul Lewis from the University of Connecticut, Mark Holder from the University of Kansas, Peter Beerli from Florida State University, and David Swofford from Duke University.

I would especially like to thank my colleagues, both past and present, at the Laboratory of Molecular Evolution. I have been lucky to have been part of an excellent research group and would like to thank Adam Bazinet and Matt Conte for their companionship both in and outside the laboratory.

I would also like to express my gratitude to the staff at the Department of Biology and at the University of Maryland Institute for Advanced Computer Studies, for all the help during the years.

Thank you to my mother Deborah and my late father José Márcio for always being there for me and encouraging my pursuits. I'm also thankful to my grandfather Manuel, whom I greatly admire, for stimulating my scientific interests and for the unwavering support. My brother Lucas reviewed this thesis and his valuable comments are also gratefully acknowledged.

Finally, I express my deepest gratitude to my wife Susanna and to my daughter Lumi for their support, encouragement, and patience during the busy months of writing this thesis.

College Park, July 1, 2017

[illegible]

Daniel Ayres

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Phylogenetics

Research in evolutionary biology can generally be divided as being most closely associated with either of two broad categories:

1. *microevolution*, which involves the processes affecting changes in the genetic structure of populations; and

2. *macroevolution*, which involves the processes of speciation and extinction.

These evolutionary categories converge in that trees representing ancestor-descendent relationships are central to the conceptual and analytical framework for both micro- and macroevolution, which are embodied by population genetics and phylogenetics respectively. Although many statistical and computational inference methods are shared between these fields, the focus of this work is in advancing these inference methods specifically when applied to phylogenetic analyses.

Phylogenetic inference is a critical step in many analyses involving biological sequence data such as DNA. Modern phylogenetic analyses involve obtaining sequence data from a set of organisms, and using model-based methods to infer a binary tree. This tree represents the evolutionary history of the organisms going back to their most recent common ancestor and is, in essence, a subset of the overall tree of life. In addition to providing a basic understanding of the evolution of life, these phylogenetic relationships are very important in understanding the evolutionary dynamics, timing, and spread of many disease-causing organisms, such as viruses causing AIDS, influenza, and Ebola disease, among others.

Modern phylogenetic inference approaches use sophisticated statistical models that are very computationally demanding, often taking weeks or even months to complete. This issue has been accentuated in recent years with the increase in importance of evolutionary genomics, which involves the use of genomic, transcriptomic (RNA-Seq), or other large data sets resulting from high-throughput sequencing technologies. Evolutionary genomics data sets are typically on the order of 10^5 to 10^6 sequence positions in length and allow researchers to expand the typical number of characters that can be used in phylogenetic reconstruction from a few hundred to tens of thousands. Whereas this results in reduced estimation errors and higher confidence in inferred phylogenies [1], it also demands even more computational capacity and makes the phylogenetic inference step a significant bottleneck for many evolutionary studies.

1.1.1 Inference Software

The most effective software packages for phylogenetic inference involve either maximum likelihood (ML) estimation or Bayesian analysis. For ML estimation, arguably the most efficient is the Genetic Algorithm for Rapid Likelihood Inference

(GARLI) package. It is an open-source phylogenetic inference program that uses the ML criterion. Developed by Zwickl [2], it is loosely based on earlier genetic algorithm work by Lewis [3]. GARLI uses a genetic algorithm to search for the likelihood optimum in the joint space of tree topologies, branch lengths and model parameter values. The program was developed with the goal of increasing both the speed of ML inference and the size of the datasets that can be reasonably analyzed. This is achieved primarily through algorithmic techniques that allow for accurate discrimination between trees while performing only a small fraction of the computation required by older methods.

For Bayesian phylogenetic analysis, MrBayes is arguably the most successful software package. Developed primarily by Ronquist and Huelsenbeck [4], it uses a Markov Chain Monte Carlo (MCMC) method for sampling from the posterior probability distribution in a stepwise fashion. Each new step is either accepted or rejected based on the change in likelihood, and the posterior probability for each parameter value is proportional to the frequency with which that value is observed.

Another software tool for evolutionary inference that applies a Bayesian approach is the Bayesian Evolutionary Analysis Sampling Trees (BEAST) program [5]. While it is oriented towards phylogeny dating, it uses the same core statistical methods as MrBayes and has similar computational algorithms.

Although the software packages described above are very effective and efficient, more complex analyses can take hundreds or even thousands of hours. These tools share the same computational bottleneck, which is the calculation of the likelihood of the sequence data given a proposed tree.

1.1.2 Likelihood Function

The most effective methods for phylogenetic inference, both for ML estimation and Bayesian analyses, involve computing the probability of observed sequence data for a set of taxa given an evolutionary model and phylogenetic tree, which is often referred to as the (observed data) likelihood of that tree. Felsenstein demonstrated an algorithm to calculate this probability [6], and his algorithm recursively computes partial likelihoods via simple sums and products. These partial likelihoods track the probability of the observed data descended from an internal node conditional on a particular state at that internal node.

The likelihood calculations apply to a subtree comprising a parent node, k, two child nodes, ℓ and m, and connecting branches of length, t_ℓ and t_m (Fig. 1.1 on the following page). It is repeated for all such subtrees within the larger tree being considered. This partial likelihood function is as follows [6]:

$$L_k^{(i)}(z) = \left(\sum_x \Pr(x|z, t_\ell) L_\ell^{(i)}(x)\right) \times \left(\sum_y \Pr(y|z, t_m) L_m^{(i)}(y)\right) \tag{1.1}$$

This calculation is repeated for each character i in the data (in the form of a multiple sequence alignment), for each state z that a character can assume (e.g., adenine, cytosine, guanine, or thymine, for a nucleotide model sequence), and for each internal node in the proposed tree. The computational complexity of the likelihood calculation for a given tree is $O(p \times s^2 \times n)$, where p is the number of patterns in the sequence (typically on the order of 10^2 to 10^6), s is the number

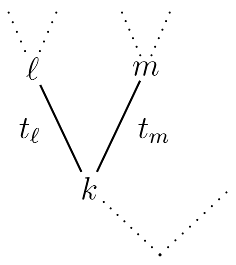


Figure 1.1: Likelihood subtree to which the partial likelihood calculation applies. Solid lines depict focus subtree branches, dotted lines contextual branches.

of states each character in the sequence can assume (typically 4 for a nucleotide model, 20 for an amino-acid model, or 61 for a codon model), and n is the number of operational taxonomic units (e.g., species, alleles).

Additionally, the tree search space is very large; the number of unrooted topologies possible for n operational taxonomic units is given by the double factorial function $(2n-5)!!$ [7]. Thus, to explore even a fraction of the total search space, a very large number of topologies are evaluated, and hence a very great number of likelihood calculations have to be performed. This leads to analyses that can take days, weeks or even months to run. Further compounding the issue, rapid advances in the collection of DNA sequence data have made the limitation for biological understanding of these data an increasingly computational problem. For phylogenetic inferences, the computational bottleneck is most often the calculation of the likelihoods on a tree. Hence, speeding up the calculation of the likelihood function is key to increasing the performance of these analyses.

1.2 Parallel Computing

Recent advances in computer hardware have come primarily in the form of increasingly parallel architectures, such as wider Central Processing Unit (CPU) vectorization intrinsics (e.g., Advanced Vector Extensions), many-core CPUs (e.g., Intel Xeon Phi), and Graphics Processing Units (GPU) specifically targeting general-purpose computing (NVIDIA Tesla). Of these parallel computing solutions, modern GPUs may offer the highest performance potential if the algorithm in question can be efficiently mapped to their architecture.

1.2.1 General-Purpose Computing on GPUs

GPU hardware is a significant source of computing power that is present in nearly every modern computer. High-end GPUs can achieve over 10 TFLOPS (trillions of floating point operations per second) in single precision floating-point format, which represents nearly a ten-fold increase over the latest CPUs [8]. Additionally, GPU performance has sustained a significantly greater rate of growth when compared to CPUs [8]. Traditionally GPUs have been used for graphics processing, as their name implies. In the last decade however, general-purpose computing on graphics processing units (GPGPU) has quickly become an important area of research, and scientific computing applications have seen significant performance improvements from adaption to this hardware.

GPU organization generally follows the SIMD (single-instruction, multiple-data) model and is specialized for computationally intensive, highly parallel applications. The hardware design is such that, when compared to CPUs, many more transistors are devoted to data processing rather than data caching and flow control [8]. More specifically, GPUs are especially adapted to address problems that can be expressed as data parallel computations where the same instructions are executed on many data elements in parallel and with high arithmetic intensity (the ratio of arithmetic operations to memory operations). Each individual sequence of computation is called a *thread* and can operate on a different section of the data. Because the same instructions are executed for each data element, there is a lower requirement for sophisticated flow control.

In order to take full advantage of many-core hardware such as GPUs, problems must be carefully mapped to these architectures, as its programming model is significantly different from that of traditional CPUs. This is a non-trivial problem and one that is the basis for much of this work.

1.2.2 Concurrency

The process of adapting existing computational methods to parallel architectures benefits from a broader perspective on *concurrency*, the simultaneous execution of independent operations. The following is the abstract from a talk, *Some Thoughts About Concurrency*, given by Ivan Sutherland at the 2010 USENIX Annual Technical Conference, which highlights the importance and timely relevance of this work as well as its non-trivial character.

Our industry has grown up with a sequential model of computing, evolved *to husband the logic associated with a few vacuum tubes. Now we must struggle to harness the vast concurrency of modern transistor circuits. Is concurrency fundamentally hard, or does it just seem hard because of our history of sequential programming? I believe some of each. Concurrency is fundamentally hard for only two reasons. One is that concurrent action requires coordination. The other is that concurrent action of many processes can produce an exponential explosion of states. How can we be sure that all such states are benign?*

Concurrency is easy when we escape its details. Maybe instead of "programming sequential processes" we might better "configure concurrent communication." A communication view of computing matches well the cost structure of modern hardware, where logic is now essentially free but moving data is relatively slow and expensive in time and energy. Making communication central to computation also prepares us for the increasing role geometry will play in the future of computing. New thinking may be essential to harnessing the vast concurrency provided by modern transistor circuits.

This idea of "configuring concurrent communication" well describes an overarching theme of this research and is a timely perspective, as it is often stated that, "modern computer architectures are not getting faster, but are getting wider". Essentially by "wider" what is meant is that more concurrent operations are supported.

1.3 Concurrent Computation in Phylogenetics

For this work, the general model for configuring concurrent communication is identifying all independent computation in phylogenetic inference methods and decomposing these for parallel computation in the most efficient manner possible. This emphasis on concurrency originates from an understanding of the specific characteristics of the computational problem — computing the likelihood function (Section 1.1.2 on page 4) — and how it is used for analyses within the domain science — phylogenetics — and recognizing the opportunities presented by trends in processor design for increasing concurrency.

Below we identify concurrent computation opportunities in likelihood-based phylogenetic inference methods, categorized by granularity.

1.3.1 Fine-Grained Parallelism

1.3.1.1 Character-Level Parallelism

Each character (e.g, sequence position) in a partial likelihood array can be computed autonomously, and subsequently combined to obtain the likelihood of the tree. In practice, the decomposition of the characters may be to the individual level, or groups of characters, with the decision often made with consideration of the processing and memory transfer capacity of the hardware being employed (e.g., number of cores available, or threads efficiently supported). Because the largest proportion of computation in statistical phylogenetics is concentrated at this level via the likelihood calculation (Section 1.1.2 on page 4), parallelism at this level has been the most common focus in pursuit of improved performance. Sequence-level parallelism has been implemented on GPUs [9–15], Field Programmable Gate Arrays (FPGAs) [16–19], STI Cell Broadband Engine Architecture [9,10], multi-core CPUs using OpenMP [9,10], and CPUs using MPI [20–23].

1.3.2 Medium-Grained Parallelism

1.3.2.1 Sequence Partitions

Evolutionary analyses benefit from increases in modeling flexibility. One clear way of improving model flexibility is to allow independent estimation of model parameters for character subsets (e.g., codon positions). This is typically referred to as a partitioned model and is a technique available in all phylogenetic software packages of importance. This concurrency opportunity is similar to character-level parallelism, however it presents additional computational complexity as the likelihood of different characters may be estimated under different models.

1.3.2.2 Independent Subtrees

Another opportunity for concurrent computation of phylogenetic partial likelihoods lies in the fact that subtrees (Fig. 1.1 on page 5) may be autonomous in relation to shared descendants. This means they may be calculated independently within the larger tree of which they are parts.

The number of subtrees requiring calculation for any full tree is $n - 1$ where,

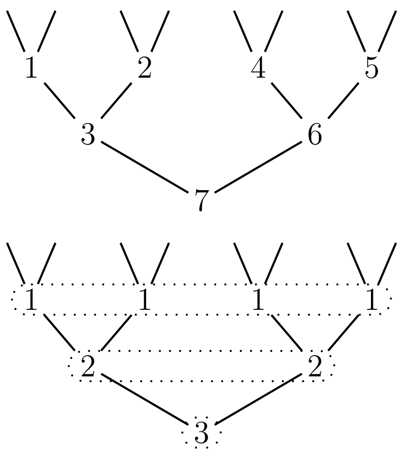


Figure 1.2: Upper: Example tree with post-order traversal and node calculations in series corresponding to node numbers, with $n-1 = 7$ node calculations. Lower: Same tree with reverse level-order, or breadth-first, traversal. Concurrent node calculations possible for independent nodes designated with the same node numbers enclosed by dotted lines, with $\lceil log_2\, n \rceil = 3$ sets.

again, n is the number of operational taxonomic units (e.g., species, alleles), which is the number of tips (leaves) on the tree. Most current phylogenetic algorithms typically use a post-order traversal when calculating tree likelihood, calculating each of the $n - 1$ subtrees in series (Fig. 1.2, upper). Often many of these subtrees are autonomous and likelihoods for each can be calculated concurrently. In the case of a fully balanced tree the number of autonomous subtrees is maximized, and reverse level-order traversal can be done in sets of concurrent operations corresponding to the number of levels in the tree, $\lceil log_2\, n \rceil$ (Fig. 1.2, lower). This exploit of tree level-group concurrency is similar to a classic parallel reduction scheme.

1.3.3 Coarse-Grained Parallelism

1.3.3.1 Independent Chains in Bayesian Analyses

Bayesian-inference analyses are often conducted as a series of alternating model parameter value proposals and comparisons of likelihoods for proposed and present states via Markov chain Monte Carlo (MCMC) using the Metropolis algorithm [24], or Hastings (Metropolis-Hastings) algorithm [25].

One common strategy for increasing the effectiveness of characterizing the posterior probability distribution and avoiding being entrapped in local optima is the Metropolis-coupled Markov chain Monte Carlo (MCMCMC) [26] algorithm. With MCMCMC multiple chains are employed, and phylogenetic applications typically use 4. One of these is the *cold* chain from which parameter values are recorded, and the others are *heated* to differing degrees and thus comprise the *hot* chains. The heating comes by analogy to simulated annealing, and here the temperature refers to what is in effect a rescaling of the likelihood to increase acceptance of proposals, and hence promote mixing (i.e., more ready exploration of the posterior). At random intervals, states of a random pair of adjacent chains (i.e., a pair of chains of sequential temperatures) are compared using an MCMC algorithm, and positions in the posterior (or alternatively the chain temperatures) are swapped if the hotter chain state (functioning as the proposal) is accepted. Some of these state swap events involve the cold chain, and hence impact the recorded values characterizing the posterior. This MCMCMC strategy is used by several important programs including MrBayes. Concurrent computation of multiple chains within MCMCMC has been implemented using message passing with MPI [20–23, 27, 28], shared memory [27, 28], and on GPUs [13, 29].

1.3.3.2 Run-Level Parallelism

There are several common use cases where replicate analyses are conducted in phylogenetics. For example, in maximum likelihood analyses independent runs might constitute multiple search replicates for the best tree using random starting points in combination with a heuristic search algorithm, or a set of bootstrap trees to estimate uncertainty in the inference [30]. This level of parallelism is relatively simple and can be done within a program, as in PAUP* [31], GARLI [2], RAxML [32], and other programs, or externally using aggregation of individual runs and post-processing to generate bootstrap values and other summary statistics in a grid computing system [33].

In Bayesian programs, entire analyses are independently run starting from random initial conditions (e.g., random starting trees). These separate analyses can be done asynchronously to check consistency of results, as suggested when doing analyses with BEAST, or synchronously to assess convergence, e.g., using Potential Scale Reduction Factor [34], as done in MrBayes. Both of these situations provide straightforward opportunities for parallel computation.

1.4 Preliminary Work

Initial work on parallel computation of phylogenetic inference methods was focused on exploring fine-grained GPU-acceleration for the program GARLI [2]. This effort was done using the CUDA (previously an acronym for Compute Unified Device Architecture) toolkit, which is a software development kit and set of programming libraries from GPU manufacturer NVIDIA [8]. As part of the work, GPU functions, or kernels, were developed for the partial likelihood calculation algorithm in GARLI.

The GPU functions developed in this preliminary work parallelized the work across threads. Each thread computed the likelihood for a different combination of sequence position and character state. These kernels supported calculation using nucleotide, amino acid, and codon models. Speedups of over 70× were achieved when the kernels were run in isolation. However, the initial method of integration with the program imposed severe memory transfer penalties and the speedups dropped by several fold when running the full tree search algorithm (Fig. 1.3 on the following page).

1.5 The BEAGLE library

Contemporaneously to our initial investigations into parallel algorithms for phylogenetic inference with GARLI, researchers working with BEAST [5] achieved similarly impressive results [11]. Given these initial results and in order to benefit the phylogenetic inference community more broadly, we collaborated to further develop

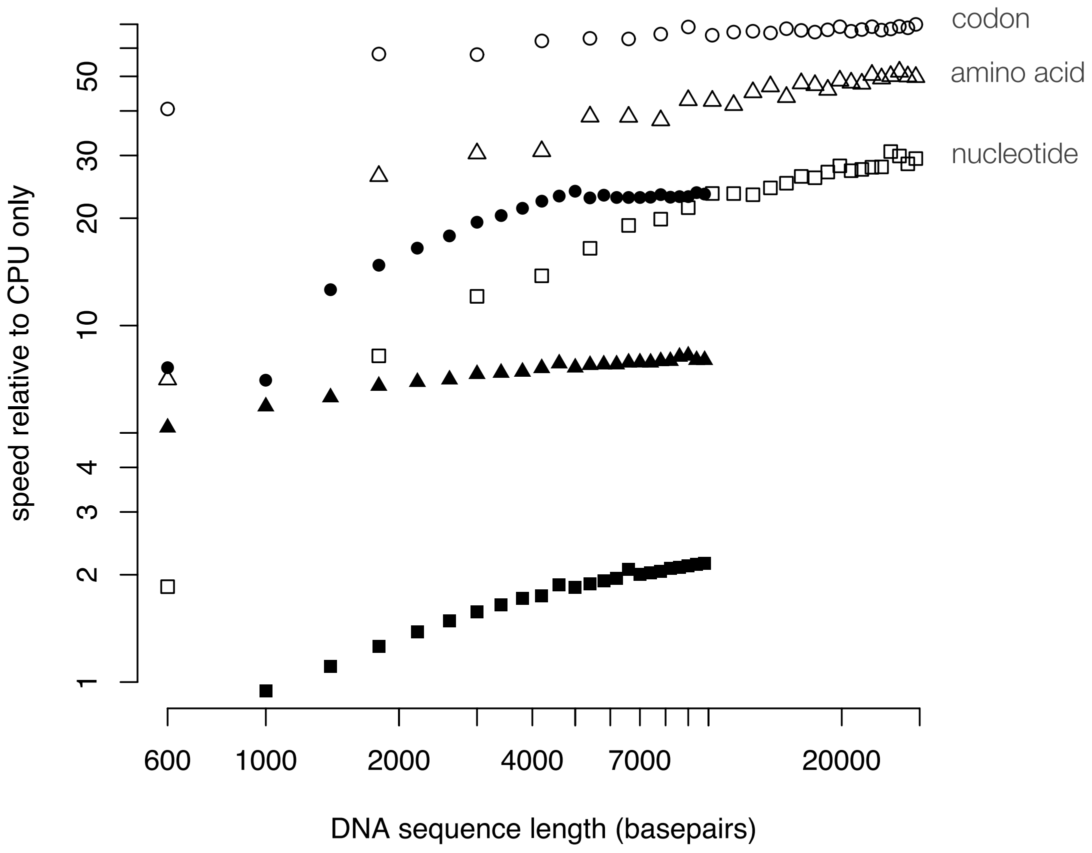


Figure 1.3: Speedups for likelihood computation with GARLI on an NVIDIA GTX 280 GPU, relative to serial CPU version running on an Intel Core i7 Nehalem processor, and under different sequence lengths and evolutionary models. Black data points indicate performance with data transfers to host (CPU) memory included.

this work as part of a software library that could be used by any program in the field.

The realization of this endeavour is the Broad-platform Evolutionary Analysis General Likelihood Evaluator (BEAGLE) [12] (Chapter 2 on page 29), a high-performance likelihood-calculation platform for phylogenetic applications. BEAGLE defines a uniform application programming interface (API) and includes a collection of efficient implementations for evaluating likelihoods under a wide range of evolutionary models, on GPUs as well as on multi-core CPUs. The BEAGLE library can be installed as a shared resource, to be used by any software aimed at phylogenetic reconstruction that supports the library. This approach allows developers of phylogenetic software to share optimizations of the core calculations and for programs that use BEAGLE to automatically benefit from the improvements to the library. For researchers, this centralization provides a single installation to take advantage of new hardware and parallelization techniques.

1.5.1 Parallel Computation with BEAGLE

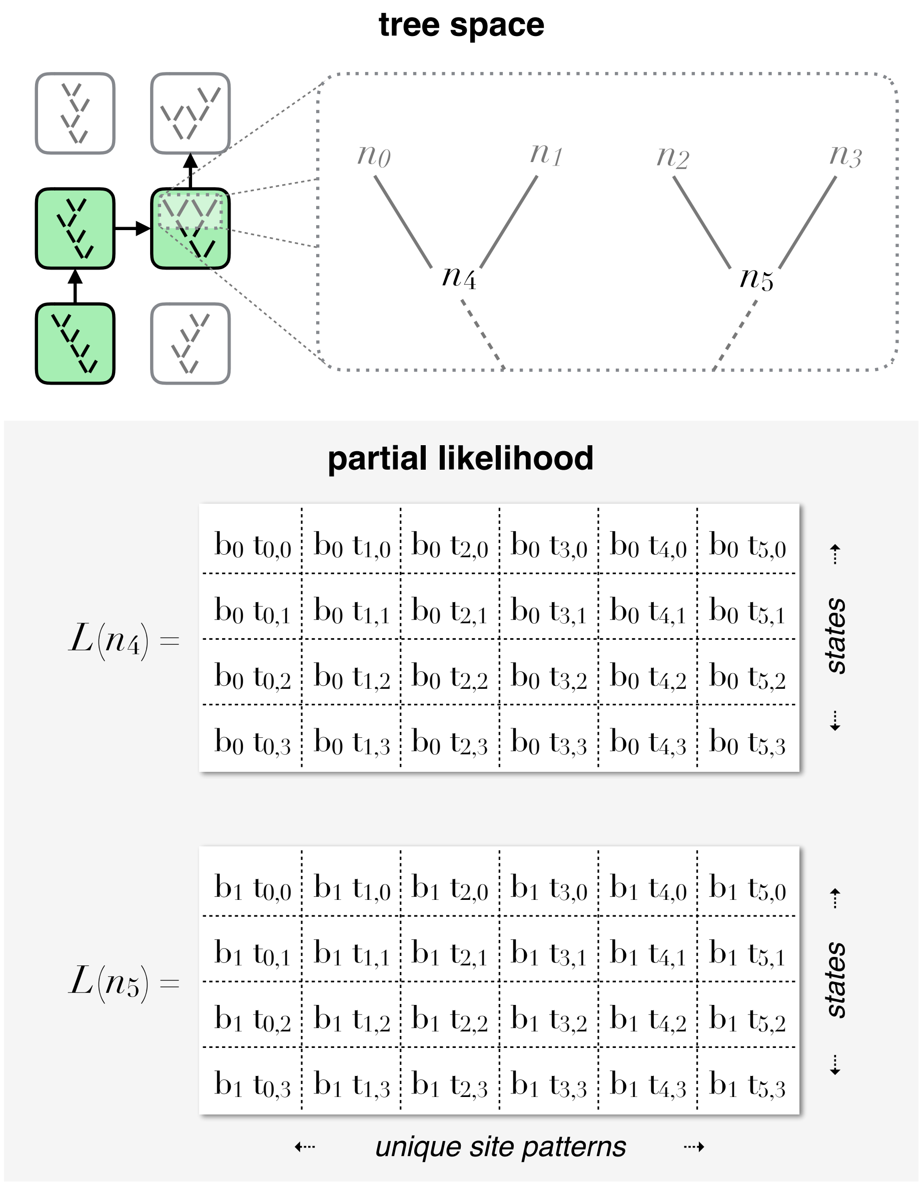
Recognizing the different levels of concurrency available in phylogenetic inference computing (Section 1.3 on page 9), as well as the different strengths of different parallel computing hardware architectures, BEAGLE implements parallelism in a variety of ways.

1.5.1.1 Fine-Grained Parallelism

BEAGLE exploits GPUs via fine-grained parallelization of functions necessary for computing the likelihood on a phylogenetic tree. Phylogenetic inference programs typically explore tree space in a sequential manner (Fig. 1.4 on the next page, *tree space*) or with a small number of sampling chains, thus offering a low upper limit for coarse-grained parallelization. In contrast, the crucial computation of partial likelihood arrays at each node of a proposed tree presents an excellent opportunity for fine-grained data parallelism, which GPUs are especially suited for. The use of many lightweight execution threads incurs very low overhead on GPUs and the presence of large numbers of positions and multiple states enables efficient parallelism at this level (Fig. 1.4 on the following page, *partial likelihood*).

Furthermore, BEAGLE uses GPUs to parallelize other functions necessary for computing the overall tree likelihood, thus minimizing data transfers between the CPU and GPU. These additional functions include those necessary for computing branch transition probabilities, for integrating root and edge likelihoods, and for summing site likelihoods.

BEAGLE also provides SSE and OpenCL implementations for exploiting fine-grained parallelism on CPUs, which vectorize likelihood calculations across characters and character states. These solutions, however, offer only a modest performance benefit as CPU vectorization intrinsics are of limited width (128 bits are available with SSE and up to 512 bits with AVX vectorization). Additionally, CPU architectures have lower memory bandwidth than GPUs and we have found this to be a



$$L(n_4) = \begin{array}{|c|c|c|c|c|c|}
\hline
b_0\ t_{0,0} & b_0\ t_{1,0} & b_0\ t_{2,0} & b_0\ t_{3,0} & b_0\ t_{4,0} & b_0\ t_{5,0} \\
\hline
b_0\ t_{0,1} & b_0\ t_{1,1} & b_0\ t_{2,1} & b_0\ t_{3,1} & b_0\ t_{4,1} & b_0\ t_{5,1} \\
\hline
b_0\ t_{0,2} & b_0\ t_{1,2} & b_0\ t_{2,2} & b_0\ t_{3,2} & b_0\ t_{4,2} & b_0\ t_{5,2} \\
\hline
b_0\ t_{0,3} & b_0\ t_{1,3} & b_0\ t_{2,3} & b_0\ t_{3,3} & b_0\ t_{4,3} & b_0\ t_{5,3} \\
\hline
\end{array}$$

$$L(n_5) = \begin{array}{|c|c|c|c|c|c|}
\hline
b_1\ t_{0,0} & b_1\ t_{1,0} & b_1\ t_{2,0} & b_1\ t_{3,0} & b_1\ t_{4,0} & b_1\ t_{5,0} \\
\hline
b_1\ t_{0,1} & b_1\ t_{1,1} & b_1\ t_{2,1} & b_1\ t_{3,1} & b_1\ t_{4,1} & b_1\ t_{5,1} \\
\hline
b_1\ t_{0,2} & b_1\ t_{1,2} & b_1\ t_{2,2} & b_1\ t_{3,2} & b_1\ t_{4,2} & b_1\ t_{5,2} \\
\hline
b_1\ t_{0,3} & b_1\ t_{1,3} & b_1\ t_{2,3} & b_1\ t_{3,3} & b_1\ t_{4,3} & b_1\ t_{5,3} \\
\hline
\end{array}$$

Figure 1.4: Diagrammatic example of the tree sampling process and medium and fine-grained parallel computation of phylogenetic partial likelihoods using BEAGLE on GPUs for a nucleotide-model problem with 5 taxa, 5 site patterns. Each entry in a partial likelihood array L is assigned to a separate GPU thread t, and each array is assigned to a separate GPU execution block b. In this simplified example, 40 GPU threads are created to enable parallel evaluation of each entry of the partial likelihood arrays $L(n_4)$ and $L(n_5)$.

limiting factor when it comes to fine-grained parallel computation of phylogenetic likelihoods.

1.5.1.2 Medium-Grained Parallelism

In order to calculate the overall likelihood of a proposed tree, phylogenetic inference programs perform a tree traversal, evaluating a partial likelihood array at each node. When using BEAGLE, the evaluation of these multi-dimensional arrays is offloaded to the library. When partial likelihood arrays are indepedent from one another, they are also evaluated in parallel, with BEAGLE assigning the calculation of each array to separate execution blocks on the GPU (Fig. 1.4 on the previous page, *partial likelihood*).

For partitioned analyses, BEAGLE can also exploit multi-core CPUs and GPUs by parallelizing the computation of multiple data subsets [35,36] (Chapters 4 on page 51 and 5 on page 84). This cabapility suits the trend of increasingly large molecular sequence data sets, which are often heavily partitioned in order to better model the underlying evolutionary processes.

1.5.1.3 Coarse-Grained Parallelism

Phylogenetic inference programs which implement multiple Bayesian chains or independent runs can invoke multiple BEAGLE library instances, one for each run. For effective parallelism, this requires multiple hardware resources (e.g., multiple GPUs) and for each library instance to be assigned to a separate device.

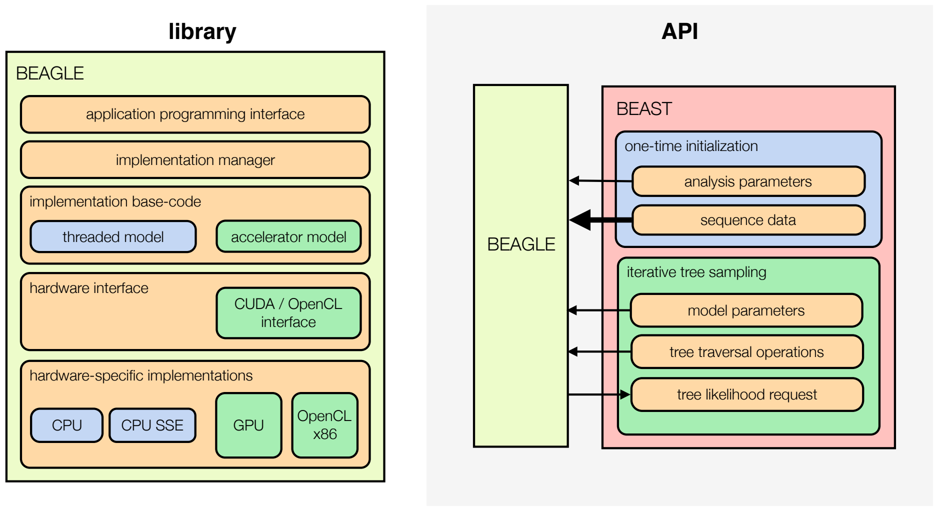


Figure 1.5: Layer diagrams depicting BEAGLE library organization, and illustration of API use. Arrows indicate direction and relative size of data transfers between the client program and library.

1.5.2 Design

1.5.2.1 Library

The general structure of the BEAGLE library can be conceptualized as a set of layers (Fig. 1.5, *library*), the uppermost of which is the application programming interface. Underlying this API is an implementation management layer, which loads the available implementations, makes them available to the client program, and passes API commands to the selected implementation.

The design of BEAGLE allows for new implementations to be developed without the need to alter the core library code or how client programs interface with the library. This architecture also includes a plugin system, which allows implementation-specific code (via shared libraries) to be loaded at runtime when the required dependencies are present. Consequently, new frameworks and hardware platforms can more easily be made available to programs that use the library, and ultimately to users performing phylogenetic analyses.

The implementations in BEAGLE derive from two general models. One is a threaded CPU implementation model, which does not directly use external frameworks. Under this model, there is a parallel CPU implementation, and one with added SSE intrinsics, which uses vector processing extensions present in many CPUs to further parallelize computation across character state values.

The other implementation model involves an explicit parallel accelerator programming model, and uses the CUDA and the OpenCL external computing frameworks to exploit parallel hardware [35] (Chapter 4 on page 51). It implements fine-grained parallelism for evaluating likelihoods under arbitrary molecular evolutionary models, thus harnessing the large number of processing cores to efficiently perform calculations [11, 12]. This parallel implementation model communicates with the CUDA and OpenCL APIs through a single internal interface.

Further significant sharing of code between CUDA and OpenCL exists at the kernel level. There is a single set of kernels for both frameworks, with keywords for each being defined at the pre-processor stage. Though there is a common kernel code-base for both frameworks, functions that impart a crucial effect on performance are differentiated for each hardware type. This allows for distinctly optimized parallel implementations that are shown in Figure 1.5, one for NVIDIA and OpenCL-compatible GPUs and one for OpenCL-compatible x86 parallel resources such as multicore CPUs with SIMD-extensions.

1.5.2.2 Application Programming Interface

The BEAGLE API was designed to increase performance via parallelization while reducing data transfer and memory copy overhead to an external hardware accelerator device (e.g., GPU). Client programs, such as BEAST [5], use the API to offload the evaluation of tree likelihoods to the BEAGLE library (Fig. 1.5 on page 20, *API*). API functions can be subdivided into two categories: those which are only executed once per inference run and those which are repeatedly called as part of an iterative sampling process. For the one-time initialization process, client programs use the API to indicate analysis parameters such as tree size and sequence length, as well as specifying the type of evolutionary model and hardware resource(s) to be used. This allows BEAGLE to allocate the appropriate number and size of data buffers on device memory. Also at this initialization stage, the sequence data is specified and transferred to device memory. This costly memory operation is only performed once, thus minimizing its impact.

During the iterative tree sampling procedure, client programs use the API to specify changes to the evolutionary model and instruct a series of partial likelihood operations that traverse the proposed tree in order to find its overall likelihood. BEAGLE efficiently computes these operations and makes the overall tree likelihood as well as per-site likelihoods available via another API call.

1.5.3 Performance

Peak performance with BEAGLE is achieved when using a high-end GPU. However, the relative gain over using a CPU depends on model type and problem size as more demanding analyses allow for better utilization of GPU cores. Figure 1.6 on the next page shows speedups relative to serial CPU code when using BEAGLE with an NVIDIA Tesla P100 GPU for the critical partial likelihood function, with increasing unique site pattern counts and for two model types. Computing these likelihoods typically accounts for over 90% of the total execution time for phylogenetic inference programs and the relationship between speedups and problem size observed here primarily matches what would be observed for a full analysis.

Figure 1.6 on the following page includes performance results for computing partial likelihoods under both nucleotide and codon models. The vertical axis shows the speedup relative to the average performance of a baseline serial, single threaded and non-vectorized, CPU implementation. This non-parallel CPU implementation provides a consistent performance level across different problem sizes and provides a relevant point of comparison as most phylogenetic inference software packages use serial code as their standard.

Using a nucleotide model, relative GPU performance over the CPU strongly scales with the number of site patterns. For very small numbers of patterns the GPU exhibits poor performance due to greater execution overhead relative to overall problem size. GPU performance improves quickly as the number of unique site patterns is increased and by $10,000$ patterns it is closer to a saturation point, con-

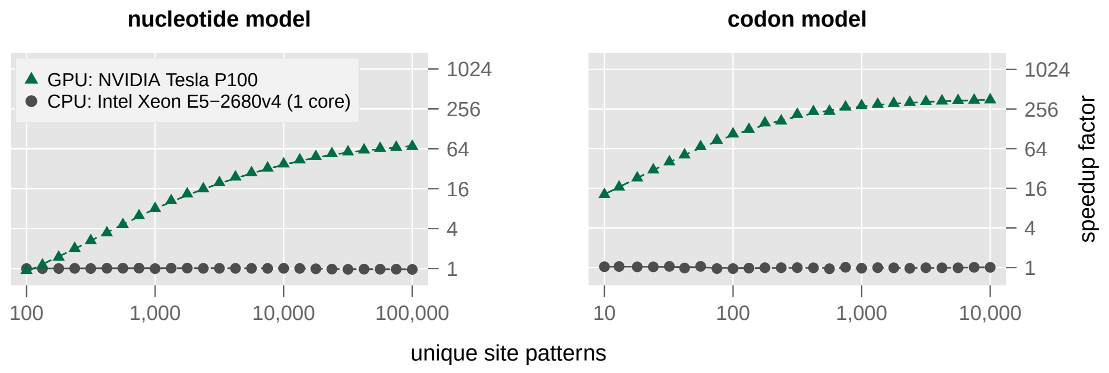


Figure 1.6: Plots showing BEAGLE partial likelihood computation performance on the GPU relative to serial CPU code, under nucleotide and codon models and for an increasing number of unique site patterns. Speedup factors are on a log-scale.

tinuing to increase but more slowly. At $100,000$ nucleotide patterns the GPU is approximately 64 times faster than the serial CPU implementation.

For codon-based models, GPU performance is less sensitive to the number of unique site patterns. This is due to the better parallelization opportunity afforded by the 61 biologically-meaningful states that can be encoded by a codon. The higher state count of codon data compared to nucleotide data increases the ratio of computation to data transfer, resulting in increased GPU performance for codon-based analyses. For a problem size with $10,000$ codon patterns the GPU is over 256 times faster than the serial CPU implementation.

1.5.4 Memory usage

When assessing the suitability of GPU acceleration via BEAGLE for a phylogenetic analysis, it is also important to consider if the GPU has sufficient on-board

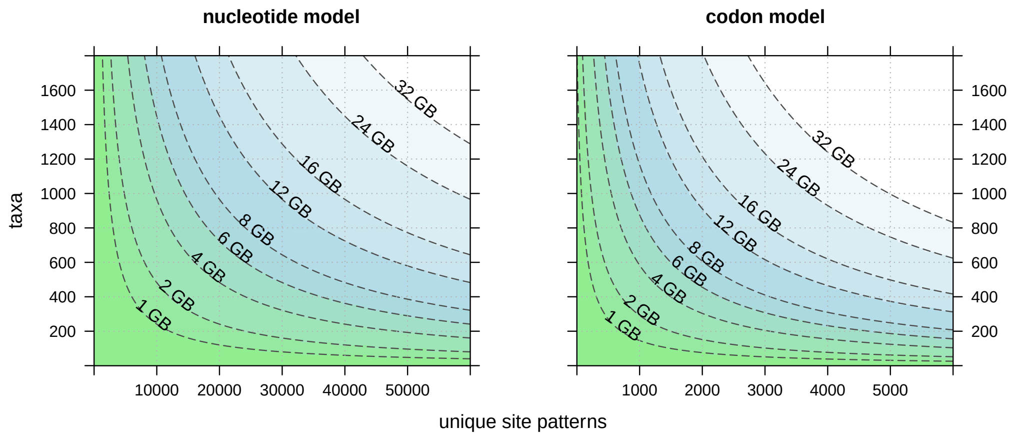


Figure 1.7: Contour plots depicting BEAGLE memory usage on GPUs for BEAST nucleotide and codon-model analyses with 4 evolutionary rate categories in double precision floating-point format, for a range of problem sizes with different numbers of taxa and of unique site patterns. Memory requirements shown here assume an unpartitioned dataset. Partitioned analyses and more sophisticated models that use multiple BEAGLE instances incur memory overhead per additional library instance.

memory for the analysis to be performed. GPUs typically have less memory than what is available to CPUs and the high transfer cost of moving data from CPU to GPU memory prevents direct use of CPU memory for GPU acceleration.

Figure 1.7 shows how much memory is required for problems of different sizes when running nucleotide and codon-model analyses in BEAST [5] with BEAGLE GPU acceleration. Note that when multiple GPUs are available, BEAST can partition a data set into separate BEAGLE instances, one for each GPU. Thus each GPU will only require as much memory as necessary for the data subset assigned to it. Typical PC-gaming GPUs have 8 GB of memory or less, while GPUs dedicated to high performance computing, such as the NVIDIA Tesla series, may have as much as 24 GB of memory.

1.5.5 Conclusion

The BEAGLE project has been very successful in bringing hardware acceleration to phylogenetics. The library has been integrated into popular phylogenetics software including BEAST [5], MrBayes [4], PhyML [37] and GARLI [2], and has been widely used across a diverse range of evolutionary studies, examples of which include: viruses, including those causing diseases such as HIV, influenza, Dengue, rabies, West Nile, and others [38–52]; strains of plague [53]; insects [54,55], snails [56], birds [57–59], fishes [60,61], lizards [62], plants [63–67]; and relationships of vertebrate globin genes [68,69].

In addition to the hundreds of researchers directly using the BEAGLE library with application programs, thousands of scientists use the BEAGLE library through software services made available via the CIPRES Science Gateway [70] that run on XSEDE resources. As is typical of users of such gateways, most are unaware that their analyses have been accelerated via the library. Furthermore the BEAGLE project has served as a reference point for similar research on parallel computation, particularly using GPUs, in evolutionary and related analyses [71–77].

The BEAGLE library is free, open-source software licensed under the Lesser GPL and available from `https://beagle-dev.github.io`.

1.6 List of Original Publications

Subsequent chapters in this thesis are based on the following original publications:

Chapter 2 Daniel L. Ayres, Aaron Darling, Derrick J. Zwickl, Peter Beerli, Mark T. Holder, Paul O. Lewis, John P. Huelsenbeck, Fredrik Ronquist, David L. Swofford, Michael P. Cummings, Andrew Rambaut, and Marc A. Suchard. BEAGLE: An application programming interface and high-performance computing library for statistical phylogenetics. *Systematic Biology*, 61(1):170–173, 2012.

Chapter 3 Fredrik Ronquist, Maxim Teslenko, Paul van der Mark, Daniel L. Ayres, Aaron Darling, Sebastian Höhna, Bret Larget, Liang Liu, Marc A. Suchard, and John P. Huelsenbeck. MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space. *Systematic biology*, 61(3):539–542, 2012.

Chapter 4 Daniel L. Ayres and Michael P. Cummings. Heterogeneous hardware support in BEAGLE, a high-performance computing library for statistical phylogenetics. In *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*, Bristol, UK, in press.

Chapter 5 Daniel L. Ayres and Michael P. Cummings. Configuring concurrent computation of phylogenetic partial likelihoods: Accelerating analyses using the BEAGLE library. In *2017 17th International Conference on Algorithms and Architectures for Parallel Processing: ICA3PP Collocated Workshops*, Helsinki, Finland, in press.

1.7 Contributions of the Author

For the publication reproduced in Chapter 2, the author contributed significantly to the design of the API and to the implementation of the library. In addition, the author devised and performed all the benchmarks, created the figures, and was the primary author of the publication.

For the publication reproduced in Chapter 3, the author contributed significantly to the integration of the BEAGLE library with MrBayes 3.2. In addition, the author contributed to the writing of the section related to this work.

For the publication reproduced in Chapter 4, the author performed all of the research and was the primary author of the publication.

For the publication reproduced in Chapter 5, the author performed all of the research and was the primary author of the publication.

Chapter 2: BEAGLE: an Application Programming Interface and High-Performance Computing Library for Statistical Phylogenetics

Most modern approaches to statistical phylogenetic inference involve computing the probability of observed character data for a set of taxa given a phylogenetic model — often a tree and continuous-time Markov chain model of character state evolution. Felsenstein demonstrated an efficient algorithm to calculate this probability [6], which is often refered to as the likelihood of the model. His algorithm recursively computes partial likelihoods via simple sums and products. These partial likelihoods track the probability of the observed data descended from an internal node conditional on a particular state at that internal node. A library that implements the calculations required by Felsenstein's algorithm is appealing because this procedure accounts for the majority of computing time in most likelihood-based phylogenetic operations. Furthermore, the algorithm offers opportunities for parallelization.

In typical phylogenetic models, likelihood calculation operations assume independence at several levels. These independencies provide the opportunity to perform operations in parallel. For example, models often assume that sites in a sequence alignment evolve independently, so that one can compute the likelihood for each site separately. The product of site-likelihoods yields the likelihood for the alignment. In models that include among-site rate variation via a finite-mixture, it is often possible to calculate conditional likelihoods given each rate category in parallel. Several other opportunities for parallelism exist at a finer-scale.

We have developed the software library BEAGLE: Broad-platform Evolutionary Analysis General Likelihood Evaluator. BEAGLE provides a uniform interface for calculating phylogenetic likelihoods under a variety of different phylogenetic models. The library implements parallelism in the likelihood calculation on important emerging computer hardware technology, including graphics processing units (GPUs) and multi-core CPUs. We intend for users to install the library as a shared resource to be used by any phylogenetic software that supports the library. This approach allows developers of phylogenetic software to share any optimizations of the core calculations and any package that uses BEAGLE will automatically benefit from the improvements to the library. For researchers, this centralization provides a single installation to take advantage of new hardware and parallelization techniques. We now describe the interface to the library and some details regarding its implementation.

2.1 Application Programming Interface (API)

2.1.1 Key Concepts

The key to BEAGLE performance lies in delivering fine-scale parallelization while minimizing data transfer and memory copy overhead. To accomplish this, the library lacks the concept or data structure for a tree, in spite of the intended use for phylogenetic analysis. Instead, BEAGLE acts directly on flexibly indexed data storage (called buffers) for observed character states and partial likelihoods. The client program can set the input buffers to reflect the data and can calculate the likelihood of a particular phylogeny by invoking likelihood calculations on the appropriate input and output buffers in the correct order. Because of this design simplicity, the library can support many different tree inference algorithms, and likelihood calculation on a variety of models. Arbitrary numbers of states can be used, as can non-reversible substitution matrices via complex eigen decompositions, and mixture models with multiple rate categories and/or multiple eigen decompositions. Finally, BEAGLE API calls can be asynchronous, allowing the calling program to implement other, coarse-scale parallelization schemes such as evaluating independent genes or running concurrent Markov chains.

2.1.2 Usage

To use the library, a client program first creates an *instance* of BEAGLE by calling `beagleCreateInstance` (further API method names can be found in the documentation distributed with the library); multiple instances per client are possible and encouraged. All additional functions are called with a reference to this instance. The client program can optionally request that an instance run on certain hardware (e.g. a GPU), or have particular features (e.g. double-precision math). Next, the client program must specify the data dimensions and specify key aspects of the phylogenetic model. Character state data is then loaded and can be in the form of discrete observed states or partial likelihoods for ambiguous characters. The observed data are usually unchanging and loaded only once at the start to minimize memory copy overhead. The character data can be compressed into unique 'site patterns' and associated weights for each. The parameters of the substitution process can then be specified, including the equilibrium state frequencies, the rates for one or more substitution rate categories and their weights, and finally the eigen decomposition for the substitution process.

In order to calculate the likelihood of a particular tree, the client program then specifies a series of integration operations that correspond to steps in Felsenstein's algorithm. Finite-time transition probabilities for each edge are loaded directly if considering a non-diagonalizable model or calculated in parallel from the eigen decomposition and edge lengths specified. This is performed within BEAGLE's memory space to minimize data transfers. A single function call will then request one or more integration operations to calculate partial likelihoods over some or all nodes. The operations are performed in the order they are provided, typically dictated by a post-order traversal of the tree topology. The client need only specify nodes for which the partial likelihoods need updating but it is up to the calling software to keep track of these dependencies. The final step in evaluating the phylogenetic model is done using an API call that yields a single log likelihood for the model given the data.

Aspects of the BEAGLE API design support both maximum likelihood (ML) and Bayesian phylogenetic tree inference. For ML inference, API calls can calculate first and second derivatives of the likelihood with respect to the lengths of edges (branches). In both cases, BEAGLE provides the ability to cache and reuse previously computed partial likelihood results, which can yield a tremendous speedup over recomputing the entire likelihood every time a new phylogenetic model is evaluated.

2.2 Methods

The core BEAGLE library is implemented in C++ with C and Java JNI interfaces. BEAGLE uses a runtime module loading system to load hardware-specific plugins (shared libraries) when suitable hardware is available. Current plugins implement BEAGLE on GPUs using CUDA and OpenCL (in development), CPUs with vector instructions using SSE, and multi-core systems via OpenMP. BEAGLE is available for Linux, Mac, and Windows operating systems, and is packaged with conventional installer methods for each.

2.2.1 GPU Implementation

The GPU implementation of BEAGLE supports both single- and double-precision arithmetic. Single-precision requires more frequent use of a rescaling scheme to avoid underflow, but allows BEAGLE to run on a greater variety of graphics processors since initial generations of such hardware did not include support for double-precision math. The GPU does fine-scale parallelization of the likelihood calculation, primarily by parallelizing across alignment sites, rate categories and state values. Models such as amino acid (20 states) or codon models (64 states), therefore permit a greater degree of parallelization than nucleotide models (4 states) and also yield the most notable speedups on GPU hardware [11]. The CUDA kernels load using the CUDA driver API, which enables them to be compiled at runtime and utilize features specific to the particular hardware and CUDA version installed. Multiple GPUs can be seamlessly utilized simultaneously via multiple BEAGLE instances.

2.2.2 CPU-based Implementations

In addition to a standard, serial CPU implementation, BEAGLE includes two other CPU-based implementations that exploit parallelism in different ways. An SSE implementation in double-precision uses vector processing extensions present in many CPUs to parallelize computation across character state values. Single-precision SSE-vectorization has not been a BEAGLE priority as other phylogenetic tools already provide this feature [31, 78] and, so, is not yet available in BEAGLE. The OpenMP implementation uses multiple threads to parallelize computation across rate categories. Although finer-scale parallelization, equivalent to that achieved for GPU devices, could be attempted it is unlikely to yield significant speedups due to the thread synchronization overhead in the OpenMP model.

2.3 Example

2.3.1 Program Speedups

Currently, three popular phylogenetic software packages interface with BEAGLE: MrBayes [78] and BEAST [79], which use Bayesian inference, and GARLI [2], which uses a ML approach. We benchmarked each of these programs to compare the speed of their native likelihood calculators to the BEAGLE implementations. In order to better exploit the parallelism offered by the GPU implementation we used a dataset with a large number of alignment sites and ran it under both nucleotide and codon models. More specifically, the dataset used had 15 taxa and 18792 nucleotide columns, 8558 of which were unique; for the codon model, 6080 of the 6264 site patterns were unique. This dataset was a subset of a larger arthropod dataset [80]. We performed these benchmarks on a standard desktop PC with a 2.9GHz Intel Core i7-930 CPU and 6 GB of 1.6 GHz DDR3 RAM. The PC was equipped with an NVIDIA GTX 580 GPU, with 1.5 GB of RAM and 512 processing cores running at 1.5 GHz.

Figure 2.1 on the following page shows run-time speedups for each program when using BEAGLE CPU, SSE, and GPU implementations under nucleotide and codon models. For the GPU implementation, we also benchmarked in single-precision

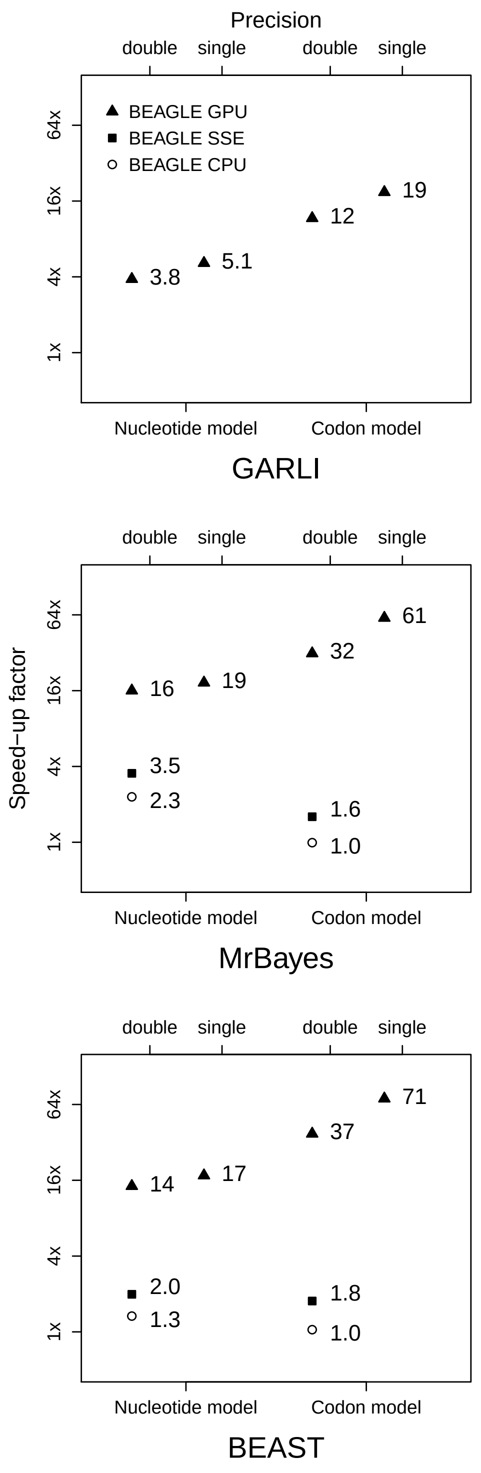


Figure 2.1: Performance using the BEAGLE library relative to the native, sequential CPU implementations of phylogenetic analysis programs GARLI, MrBayes, and BEAST. Speedup factors are on a log-scale.

mode. Reported speedups are relative to the run-time when using the native, sequential CPU implementation of each program. We note that the GARLI interface with BEAGLE is not fully optimized. While we expect that further integration work will produce positive results, in our tests only the GPU implementation achieved effective speedups. We have thus omitted the results from the CPU-based implementations.

For the BEAGLE GPU implementation we observe significant speedups across all programs. The speedups are largest under the codon models, as they allow for better utilization of the GPU cores. We also observe the higher performance cost of double-precision calculation on the GPU relative to single-precision. Overall, the highest speedup is 71-fold, for the BEAGLE GPU single-precision implementation when compared to the BEAST native implementation, under the codon model.

We note that not every analysis run on a GPU will achieve the same speedups we report and, in some circumstances, using the BEAGLE GPU implementation may result in a slower overall runtime than using a CPU implementation. Several factors affect the relative performance. Beyond state-space size and numerical precision, the number of unique alignment columns and the hardware specifications of the GPU, especially numbers of cores and memory bandwidth, are important factors. We recommend that users first assess the relative performance of the GPU implementation with their setup by performing short comparative runs, which specify a smaller chain length or fewer generations.

2.4 Conclusion

BEAGLE is an API and library for high performance evaluation of phylogenetic likelihoods. The API provides a uniform interface for performing calculations on an expanding variety of compute hardware platforms including GPUs, multi-core CPUs and SSE vectorization. On GPUs, the library provides novel algorithms and methods for evaluating likelihoods under arbitrary molecular evolutionary models, harnessing the large number of processing cores to efficiently parallelize calculations. Current results show speedups of up to 71-fold on a single GPU over CPU-based likelihood-calculators. BEAGLE is currently integrated with three state-of-the-art phylogenetic software packages: MrBayes, BEAST and GARLI, and compatible with many more. Forthcoming extensions include OpenCL support, single-precision SSE-vectorization, improved performance for highly partitioned datasets and additional high-level language wrappers, such as Python. BEAGLE is freely available from http://beagle-lib.googlecode.com under the GNU Lesser General Public License and new collaborators are welcome.

Funding

The work was supported by the National Science Foundation (grant numbers DBI-0755048, DEB-0732920, DEB-1036448, DMS-0931642, EF-0331495, EF-0905606, EF-0949453); the National Institutes of Health (grant numbers R01-HG006139, R01-GM037841, R01-GM078985, R01-GM086887); the Biotechnology and Biological Sciences Research Council (grant number BB/H011285/1); the Wellcome Trust (grant number WT092807MA); and Google Summer of Code.

Acknowledgments

Some of the development of BEAGLE occurred at meetings at the University of Maryland, the National Evolutionary Synthesis Center (NESCent) through a working group (Software for Bayesian Evolutionary Analysis), the Mathematical Biosciences Institute at The Ohio State University, and several instances of the Workshop on Molecular Evolution (Europe and North America). We thank Jerome Regier for providing the data set used in evaluating the programs.

Chapter 3: MrBayes 3.2: Efficient Bayesian Phylogenetic Inference and Model Choice Across a Large Model Space

3.1 Overview

Bayesian Markov chain Monte Carlo (MCMC) methods quickly gained in popularity after they were introduced in statistical phylogenetics in the late 1990's [81–84]. This was due to the inherent advantages of the approach but also to the availability of easy-to-use software packages, such as MrBayes [85]. Originally, MrBayes only supported simple phylogenetic models, but the model space expanded considerably in version 3.0 [78]. In addition to a wide range of models on binary, "standard" (morphology), nucleotide and amino acid data, version 3.0 also supported mixed models. The latter allow different data partitions to be combined in the same model, with parameters linked or unlinked across partitions according to user specifications. MrBayes 3.0 was apparently the first statistical phylogenetics package to support such models [86].

Bayesian phylogenetic inference using MCMC has developed in leaps and bounds since the release of MrBayes 3.0. In particular, the relative ease with which complex models can be tackled using the MCMC machinery has led to an explosion in the development of probabilistic evolutionary models (for a review, see [87]). We have also seen the appearance of better MCMC algorithms and more sophisticated convergence diagnostics for phylogenetic models, and methods for Bayesian model choice have improved considerably.

With this note, we announce the official release of version 3.2 of MrBayes. Version 3.2 was originally intended as a relatively modest expansion of version 3.1, which added convergence diagnostics to the original features in version 3.0. Over the years, however, a number of significant new features were added to version 3.2, and large parts of the program were rewritten. When we now officially release version 3.2, it is every bit as significant in the evolution of the program as the release of version 3.0 almost a decade ago.

3.2 Description of New Features

3.2.1 Convergence

The phylogenetics community has come to accept as good practice that Bayesian MCMC results be accompanied by a critical assessment of convergence. Arguably, the best way of accomplishing this is to compare samples obtained from independent MCMC analyses. It is typically the tree samples that are most divergent in phylogenetic analyses, and we therefore introduced the average standard deviation of split frequencies (ASDSF) in MrBayes to allow quantitative assessment of the similarity among such samples.

ASDSF is calculated by comparing split or clade frequencies across multiple independent MCMC runs that ideally should be started from different randomly chosen starting trees [88]. ASDSF should approach 0.0 as runs converge to the same distribution. The frequencies of rare splits or clades are difficult to estimate accurately and these groupings are usually of marginal interest. Therefore, it may be advantageous to exclude them from the diagnostic. MrBayes allows the user to set a cutoff frequency (default value 0.10); all splits or clades occurring minimally at that frequency in at least one of the runs will be incorporated in the ASDSF.

To allow users to monitor MCMC progress, MrBayes can run several analyses in parallel and report the average (ASDSF) or maximum standard deviation of split frequencies at regular intervals. More detailed diagnostics can be obtained using the “sump” and “sumt” commands after the run has completed. They include ASDSF across runs for each of the sampled clades in addition to the potential scale reduction factor (PSRF [34]) for branch lengths, node times, and substitution model parameters. PSRF compares the variance within and between runs and should approach 1.0 as runs converge. MrBayes 3.2 also reports the effective sample size, widely used for single-run convergence diagnostics.

MrBayes 3.2 also introduces several new features intended to improve MCMC convergence rates. A number of new tree proposal mechanisms have been added, including subtree-swapping moves and extending subtree-pruning-and-regrafting moves, and the default mix of proposals has been optimized [88]. MrBayes 3.2 further includes a completely new type of tree proposal that is guided using parsimony scores. The details of the parsimony-biased proposals will be presented elsewhere; however, tentative empirical results show that they can improve the speed of convergence by an order of magnitude on some problems (see also [89]). For nontree proposals, MrBayes 3.2 implements auto-tuning that automatically adjusts tuning parameters such that a target acceptance frequency is reached [90]. Since previous versions, MrBayes supports Metropolis coupling (heated chains) to accelerate convergence. To simplify monitoring of convergence, MrBayes 3.2 prints ASDSF values, acceptance rates of moves, and acceptance rates of swaps between Metropolis-coupled chains to a separate file with a ".mcmc" suffix during runs.

3.2.2 Faster and More Convenient Computation

Much of the computational effort in a phylogenetic MCMC analysis is spent calculating likelihoods. To improve speed, MrBayes 3.2 now employs streaming single-instruction-multiple-data extensions (SSE) for all likelihood calculations. SSE instructions are supported by most current CPUs and provide low-level parallelization of arithmetic operations. Importantly, MrBayes 3.2 also supports the use of the BEAGLE library for likelihood calculations [12]. With BEAGLE, the likelihood calculations can be farmed out to one or more graphics processing units (GPUs) on compatible hardware, resulting in significant speedups for codon and amino acid models in particular. BEAGLE can also be used for likelihood computation on the CPU.

MrBayes 3.2 does not support multithreading, but it does implement the message passing interface (MPI) for efficient parallel processing across large computer clusters [28]. On many hardware platforms, including Mac OS and Linux, it is possible to use the MPI-enabled Unix version of MrBayes to take advantage of multiple cores. However, MPI parallelization is across chains, which means that the maximum number of cores or processors that can be used by MrBayes is the same as the total number of heated and nonheated chains across all simultaneous runs. For instance, two runs of four chains each would be maximally accelerated on a system with eight processors or cores. The MPI version can be combined with BEAGLE to further expand the opportunity for computational parallelization.

Finally, to facilitate long runs, MrBayes 3.2 implements checkpointing across all models. At a frequency determined by the user, all parameter samples are printed to a ".ckp" file. If desired, the analysis can later be restarted from the checkpoint file, and the final results will appear as if the run had never been stopped.

3.2.3 New Models

Many phylogenetic hypotheses concern the structure of the phylogenetic tree. To facilitate such analyses, MrBayes 3.2 implements three types of constraints on the tree: hard, negative, and partial. A hard constraint forces a split or clade to be present in all trees sampled in the MCMC analysis, whereas a negative constraint forces a split or clade to be absent. Unlike hard and negative constraints, a partial constraint (or backbone constraint) can leave the position of some taxa indeterminate. The indeterminate taxa are allowed to appear on either side of the specified split if the tree is unrooted, or either within or outside the specified clade if the tree is rooted. Several hard, negative, and partial constraints can be combined into complicated priors on the shape of the tree. However, constraints are either on or off; they cannot be associated with probabilities in the current version.

Unlike previous versions, MrBayes 3.2 supports relaxed clock models and dating. Three different relaxed clock models are available: the Compound Poisson Process (CPP [91]), the ThorneKishino 2002 (TK02 [92]), and the Independent Gamma Rate (IGR [93]) models.

The CPP model is a discrete autocorrelated model, in which rate multipliers appear on the tree according to a Poisson process. The MrBayes implementation uses a lognormal distribution for the rate multipliers instead of the modified gamma distribution proposed originally [91]. It also includes novel algorithms to allow sampling across tree space since the original paper only dealt with fixed trees.

The TK02 model is a continuous autocorrelated model. In the particular version we implemented [92], the rate of a descendant node is drawn from a lognormal distribution, the mean of which is the same as the ancestral rate and the variance of which is proportional to the length of the branch (measured in expected substitutions per site at the base rate of the clock).

The IGR model is a continuous uncorrelated model. First published as the "white noise" model [93], it is similar to the uncorrelated gamma model [94] but is mathematically more elegant in that it truly lacks time structure. In the IGR model, effective branch lengths are drawn from a gamma distribution, in which the mean is the same as, and the variance proportional to, the branch length.

Dating can be achieved in MrBayes 3.2 by calibrating interior or tip nodes in the tree; calibrated interior nodes need to be associated with hard constraints to be valid. Calibration points can be either fixed or associated with uncertainty. The birthdeath prior model on clock trees has been expanded to incorporate recent progress in the understanding of the linear constant birthdeath process with complete sampling [95], with random incomplete sampling [96], or with clustered or diversified sampling [97]. The tree moves on clock and relaxed clock trees have also been improved considerably over those that were available in previous versions.

Bayesian phylogenetic inference of species trees from multiple gene trees was first accomplished in the Bayesian estimation of species trees (BEST) software using a complex computational machinery, in which MrBayes was one of the components [98, 99]. Despite later improvements to BEST, the analyses remained slow and computationally demanding. The multispecies coalescent model has now been fully integrated in MrBayes 3.2, and several of the original algorithms have been rewritten to speed up the calculations.

3.2.4 Model Averaging and Model Choice

It is standard practice today to select a substitution model for Bayesian phylogenetic inference using a priori model selection procedures [100–103]. An alternative is to use Bayesian model jumping during the MCMC simulation to integrate out the uncertainty concerning the correct substitution model [104]. The latter procedure is now implemented in MrBayes 3.2. Rather than selecting a substitution model before the analysis, the user can now sample across all 203 possible time-reversible rate matrices according to their posterior probability. The model-jumping approach is available in all models where a four-by-four nucleotide model is a component, including doublet and codon models in addition to the ordinary nucleotide models.

Bayesian model choice using Bayes factors is rapidly gaining in popularity. Since earlier versions, MrBayes has reported the harmonic mean of the likelihoods from the MCMC sample, which can be used as a rough estimate of the model likelihood from which the Bayes factor is calculated [105]. However, there are now considerably more accurate, albeit computationally more demanding, methods [106]. Of these, MrBayes 3.2 implements the recently proposed stepping stone method [107] that uses MCMC to sample from a series of so-called power posterior distributions connecting the posterior distribution with the prior distribution. The samples across these distributions are then used to estimate the model likelihood. The stepping stone algorithm in MrBayes 3.2 uses the full MCMC machinery, including convergence diagnostics and Metropolis coupling, and can be applied to any model available in the program. For instance, it can be used to test various topological hypotheses or substitution models against each other.

3.2.5 More Output Options

MrBayes 3.2 provides more extensive output options than previous versions. The user can now request sampling of site rates, site selection coefficients, site positive selection probabilities, and ancestral states of particular nodes. A wide range of tree statistics, including the mean and variance of split or clade frequencies, node times, and branch rates, are now added as annotations to the consensus tree by the “sumt” command and can be displayed using FigTree and compatible tree viewers.

3.3 Benchmark and Biological Examples

Benchmark data on the GPU-accelerated code are provided by [12]. A number of example data sets are distributed with the program, and tutorials illustrating most of the new features are included in the program manual. Many of the dating features in MrBayes 3.2 are discussed in some detail and used in an empirical context in [108].

3.4 Availability

MrBayes 3.2 is freely available under the GNU General Public License version 3.0. The program web site (http://www.mrbayes.net) provides download links to both source code for compilation on Unix systems and to convenient installers for Windows and Mac OS systems. The installers include both MrBayes and the required BEAGLE libraries, but the BEAGLE libraries can also be installed separately using the BEAGLE installer, available at http://beagle-lib.googlecode.com. The program comes with a manual and example files. Further help is available on the program web site, which also provides instructions for reporting bugs and signing up for the MrBayes e-mail list. Instructions for accessing the MrBayes source code repository can be found at http://sourceforge.net/projects/mrbayes/develop.

Funding

The development of version 3.2 of MrBayes would not have been possible without generous support from the Swedish Research Council [2008-5629 to F.R.]; the National Institutes of Health [GM-069801 to J.P.H. and GM-086887, HG-006139 to M.A.S.]; and the National Science Foundation [DEB-0445453 to J.P.H., DEB-0949121 and DEB-0936214 to B.L., and DBI-0755048 to D.L.A.]. Incorporation of the BEST algorithms and support for the BEAGLE library was greatly facilitated by a workshop in October 2010 sponsored by the Mathematical Biosciences Institute at Ohio State University [NSF-DMS-0931642], hosted by Dennis Pearl and Marty Golubitsky.

Acknowledgments

F.R., with the assistance of M.T. and P.v.d.M., did most of the programming for version 3.2, whereas J.P.H., assisted by F.R., was responsible for the software architecture and initial code base. D.L.A., A.D., and M.A.S. helped with the BEAGLE integration and the related performance testing. L.L. assisted in the incorporation of the BEST algorithms, whereas B.L. and S.H. contributed to the implementation of particular models. We would like to thank Chris Anderson for additional assistance with the BEST algorithms. We would also like to express our deep gratitude to the many MrBayes users, who have generously contributed to the project by submitting bug reports, bug fixes, feature requests, and other comments on the software. David

Posada, Leonardo Martins, and Jeremy Brown provided constructive criticism that helped improve the manuscript.

Chapter 4: Heterogeneous Hardware Support in BEAGLE, a High-Performance Computing Library for Statistical Phylogenetics

4.1 Introduction

Advances in computer hardware, specifically in parallel architectures, such as multicore CPUs, manycore CPUs (e.g., Intel Xeon Phi), GPUs, and CPU intrinsics (e.g., SSE, AVX), have created opportunities for new approaches to computationally intensive analysis methods. The design and development process required to take advantage of these parallel computing resources begins with decisions of what hardware to support and development frameworks to use. These initial decisions typically have implications that generally constrain applicable hardware used by the software developed, as well as the complexity of the software itself. Here we describe the software design and optimization approaches used to extend the range of hardware devices supported in the upcoming release of BEAGLE, a high-performance library for statistical phylogenetics [12]. We then explore the performance of the library on a variety of modern hardware resources and platforms.

Our design harnesses parallel hardware via multiple frameworks, and includes a model for sharing kernels for CUDA and OpenCL frameworks. Our own motivations for pursuing a development plan involving multiple frameworks comprise a number of elements. First among these is our desire to serve a large community of evolutionary biologists and others doing very common, but very computationally intensive calculations. This community has access to a broad range of hardware, and developing with multiple frameworks helps our library work across this range of hardware. Secondly, use of multiple frameworks diversifies risk across both hardware and software platforms. Market forces largely determine the composition of the hardware-software ecosystem, and correctly choosing the more important among the possible combinations can be difficult in the early phases of the hype cycle and in the presence of vendor marketing. Poor choice of target hardware or development framework can result in greatly diminished impact and a poorly served domain science community.

Regardless of high apparent promise of any particular option, at least initially, diversifying across processor architectures and development frameworks seems prudent. As examples of risk in the hardware realm consider the history of processors such as the Intel Itanium and STI Cell Broadband Engine, or the current status of the Intel Xeon Phi. In the realm of frameworks illustrative examples of risk include OpenCL for Apple macOS, for which not all features are supported, and OpenCL for Xeon Phi (Knights Landing), which is not available at the time of this writing. In addition to risk reduction, diversifying across processor architectures and development frameworks results in deeper understanding of hardware features and programming approaches, which can subsequently lead to better performance across implementations.

We continue this paper by providing an abbreviated review of related work, some context of the basic problem from the application domain science and computational perspectives, a general overview of the BEAGLE library, and follow with details regarding our shared framework strategy, including various design issues, hardware-specific optimizations, and performance results.

4.2 Related Work

We restrict our abbreviated consideration of related work involving to that involving the CUDA and OpenCL frameworks, as these comprise the most widely used for GPU programming, which is a special, though not exclusive, focus of the BEAGLE library. Furthermore, it is the users of the CUDA and OpenCL frameworks who are most likely to find some of our design decisions most applicable to their own efforts. This related work can be generally classified as translators, where the objective is to take code associated with one framework, most commonly CUDA for NVIDIA GPUs, and translate to another framework or processor architecture. These translators differ in the starting code for translation, as well as the target framework or processor architecture.

Starting with the pseudo-assembly code Parallel Thread Execution (PTX) generated in the CUDA framework, Ocelot [109] targets x86 and STI Cell Broadband Engine processors, whereas Caracal [110] targets the AMD Compute Abstraction Layer (CAL), a low-level access software development layer. Source-to-source translation from CUDA has been an approach followed by others. Among the direct source code translation approach are MCUDA [111] targeting x86-based processors, and CU2CL (CUDA-to-OpenCL) [112,113], which targets OpenCL. Swan [114] also targets OpenCL, but requires the developer to replace CUDA API calls with intermediary equivalent calls in a Swan-specific syntax, and these, in turn, are translated to generate the OpenCL code.

Our own work described in this paper differs fundamentally from the work mentioned above. First, our approach is to design and develop kernels that are shared between CUDA and OpenCL, rather than to create or employ a tool for after-the-fact translation to a different framework or architecture. From our perspective and objectives this fundamental difference has several advantages over translation particularly in the areas of efficiency and simplicity: i) ready sharing of core algorithms; ii) easier accommodation of new analytical models; iii) no dependence on translators, which may or may not be up-to-date with respect to the latest framework versions, thus eliminating another potential development risk; iv) reduces duplicated code; and v) adroitly facilitates hardware-specific optimizations.

4.3 Evolutionary Biology, the Scientific Domain

Research in evolutionary biology can generally be divided as being most closely associated with either of two broad categories: i) *macroevolution*, which involves the processes of speciation and extinction; and ii) *microevolution*, which involves the processes affecting changes in the genetic structure of populations. These evolutionary categories converge in that trees representing ancestor-descendent relationships are central to the conceptual and analytical framework for both macro- and microevolution, which are embodied by phylogenetics and population genetics respectively.

In a broad sense phylogenetics is the study of evolutionary relationships. Typically, modern phylogenetic analyses involve obtaining DNA sequence data from a set of organisms, and using model-based methods to infer a binary tree. This tree represents the evolutionary history of the organisms going back to their most recent common ancestor and is, in essence, a subset of the overall tree of life.

4.3.1 Likelihood Function

The most effective methods for inferring both phylogenetic trees and gene genealogies are based on either maximum likelihood estimation or Bayesian analysis, which share the same computational bottleneck: calculation of the likelihood of trees [6]. When profiling GARLI [2], a leading phylogenetic inference program, we have observed that, for DNA models, likelihood related calculations typically constitute over 94% of the overall runtime. For more complex models (e.g., amino-acid or codon-based), likelihood calculation will typically incur an even greater proportion of the analysis time. Speeding the calculation of the likelihood function is key to increasing the performance of statistical inference-based phylogenetic analyses.

The core likelihood calculations apply to a subtree comprising a node (x_0) and its two descendant nodes (x_1 and x_2), and the connecting branches (of length t_1 and t_2), and is repeated for all such subtrees of the larger tree being considered. This partial-likelihoods function [6] is as follows.

$$L(x_0) = \left(\sum_{x_1} \Pr(x_1|x_0, t_1) L(x_1) \right) \times \left(\sum_{x_2} \Pr(x_2|x_0, t_2) L(x_2) \right) \tag{4.1}$$

This calculation is repeated for each site (i.e., sequence position), and for each possible character a site can assume (e.g., `a`, `c`, `g`, and `t`, for a nucleotide model sequence). The computational complexity of the likelihood calculation for a given tree is $O(p \times s^2 \times n)$, where p is the number of positions in the sequence (typically on the order of 10^2 to 10^6), s is the number of states each character in the sequence can assume (typically 4 for a DNA model, 20 for an amino-acid model, or 61 for a codon model), and n is the number of operational taxonomic units (e.g., species, alleles).

Thus, to explore even a fraction of the total search space, a very large number of topologies are evaluated, and hence a very great number of likelihood calculations have to be performed. This leads to analyses that can take days, weeks or even months to run. Further compounding the issue, rapid advances in the collection of DNA sequence data have made the limitation for biological understanding of these data an increasingly computational problem.

The structure of the likelihood calculation, involving large numbers of positions and multiple states, as well as other characteristics, makes it a very appealing computational fit to modern parallel microarchitectures such as multi and manycore CPUs, and especially, GPUs.

4.4 BEAGLE

BEAGLE [12] is a high-performance likelihood-calculation platform for phylogenetic applications. BEAGLE defines a uniform application programming interface (API) and includes a collection of efficient implementations for calculating a variety of phylogenetic models on different hardware devices, such as graphics processing units (GPUs), Intel Xeon Phi devices, and multicore CPUs.

The BEAGLE project has been very successful in bringing hardware accelerators to phylogenetics. The library was the first to focus on high-performance computation of the phylogenetic likelihood calculation via fine-scale parallelization. It is the most widely adopted library for this purpose and has been integrated into popular phylogenetics software including BEAST [5], MrBayes [4], and PhyML [37], and has been widely used for phylogenetic analyses. Recent work on the BEAGLE library identifying independent likelihood estimates in analyses of partitioned datasets and in proposed tree topologies, and configuring concurrent computation of these likelihoods via CUDA and OpenCL frameworks results in substantially increased performance [36].

Other proposals have been made to bring hardware acceleration to statistical phylogenetics, however these have typically focused only on MrBayes and have only applied to a subset of models the program supports [13], or have not made the source code or binaries available [77].

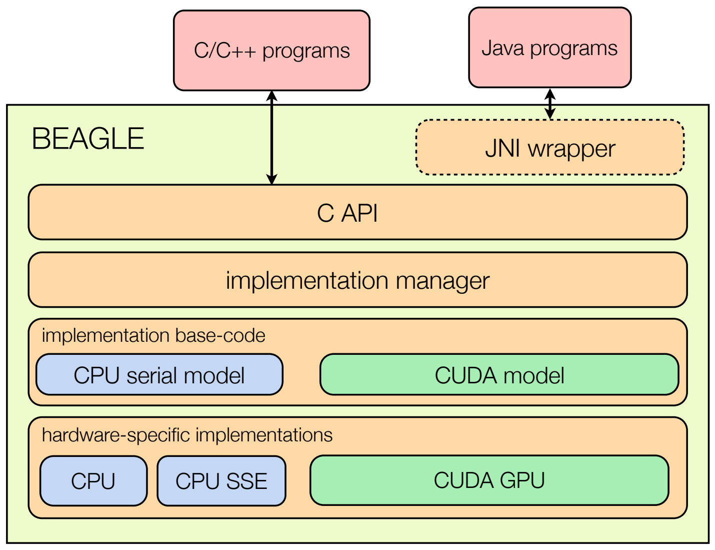


Figure 4.1: Layer diagram depicting the overall structure of the BEAGLE library version 1.

4.4.1 Overall Design

The general structure of the BEAGLE library version 1 can be conceptualized as layers (Fig. 4.1), the upper most of which is a C API. Alternatively, Java programs can use a Java Native Interface (JNI) wrapper, which is provided with the source code.

Underlying the API is an implementation management layer, which loads the available implementations, makes them available to the client program, and passes API commands to the selected implementation. Internally, the implementations in BEAGLE derive from two general models. One is a serial CPU implementation model that does not directly use external frameworks, and which comprises a standard CPU implementation, and one with added SSE intrinsics.

The other implementation model involves an explicit parallel accelerator programming model, which uses the CUDA external computing framework. This parallel implementation model communicates directly with the GPU via CUDA APIs.

4.4.2 Application Programming Interface

The BEAGLE API was designed to increase performance via fine-scale parallelization while reducing data transfer and memory copy overhead to an external hardware accelerator device. To accomplish this, the library lacks the concept or data structure for a tree, which provides for a more simplified implementation in application programs. Instead, BEAGLE acts directly on flexibly indexed data storage which stores the partial-likelihoods.

4.4.3 Implementation Overview

The design of BEAGLE allows for new implementations to be developed without the need to alter the core library code or how client programs interface with the library. This architecture also includes a plugin system, which allows implementation-specific code (via shared libraries) to be loaded at runtime when the required dependencies are present. Consequently new frameworks and hardware platforms can more easily be made available to programs that use the library, and ultimately to users performing phylogenetic analyses.

4.4.4 CPU Implementations

BEAGLE version 1 includes a serial CPU implementation, as well as an SSE implementation for nucleotide models in double-precision, which uses vector processing extensions present in many CPUs to parallelize computation across character state values.

4.4.5 CUDA Implementation

The initial version of BEAGLE exclusively used the CUDA platform to exploit NVIDIA GPUs. It implemented novel computational methods for evaluating likelihoods under arbitrary molecular evolutionary models, harnessing the large number of processing cores to efficiently parallelize calculations [11,12]. We originally chose to develop with the CUDA Driver API rather than the Runtime API due to its greater flexibility. This also facilitated sharing code with the subsequently developed OpenCL solution.

4.4.6 Parallel Computation

BEAGLE exploits GPUs via fine-grained parallelization of functions necessary for computing the likelihood on a phylogenetic tree. Phylogenetic inference programs typically explore tree space in a sequential manner (Fig. 4.2 on page 62, *tree space*) or with only a small number of sampling chains, offering limited opportunity for task-level parallelization. In contrast, the crucial computation of partial likelihood arrays at each node of a proposed tree presents an excellent opportunity for fine-grained data parallelism, for which GPUs are especially suited.

In order to calculate the overall likelihood of a proposed tree, phylogenetic inference programs perform a post-order traversal, evaluating a partial likelihood array at each node. When using BEAGLE, the evaluation of these multi-dimensional arrays is offloaded to the library. Though each partial likelihood array is still evaluated in series, BEAGLE assigns the calculation of the array entries to separate GPU threads, for computation in parallel (Fig. 4.2 on the next page, *partial likelihood*). Further, BEAGLE uses GPUs to parallelize other functions necessary for computing the overall tree likelihood, thus minimizing data transfers between the CPU and GPU. These additional functions include those necessary for computing branch transition probabilities, for integrating root and edge likelihoods, and for summing site likelihoods.

For exploiting CPU parallelism, BEAGLE provides an SSE implementation that vectorizes likelihood calculations. Additionally, in order to exploit multiple CPU cores, application programs running partitioned analyses can invoke multiple library instances, one for each data subset (or partition). This approach suits the trend of increasingly large molecular sequence data sets, which are often heavily partitioned in order to better model the underlying evolutionary processes.

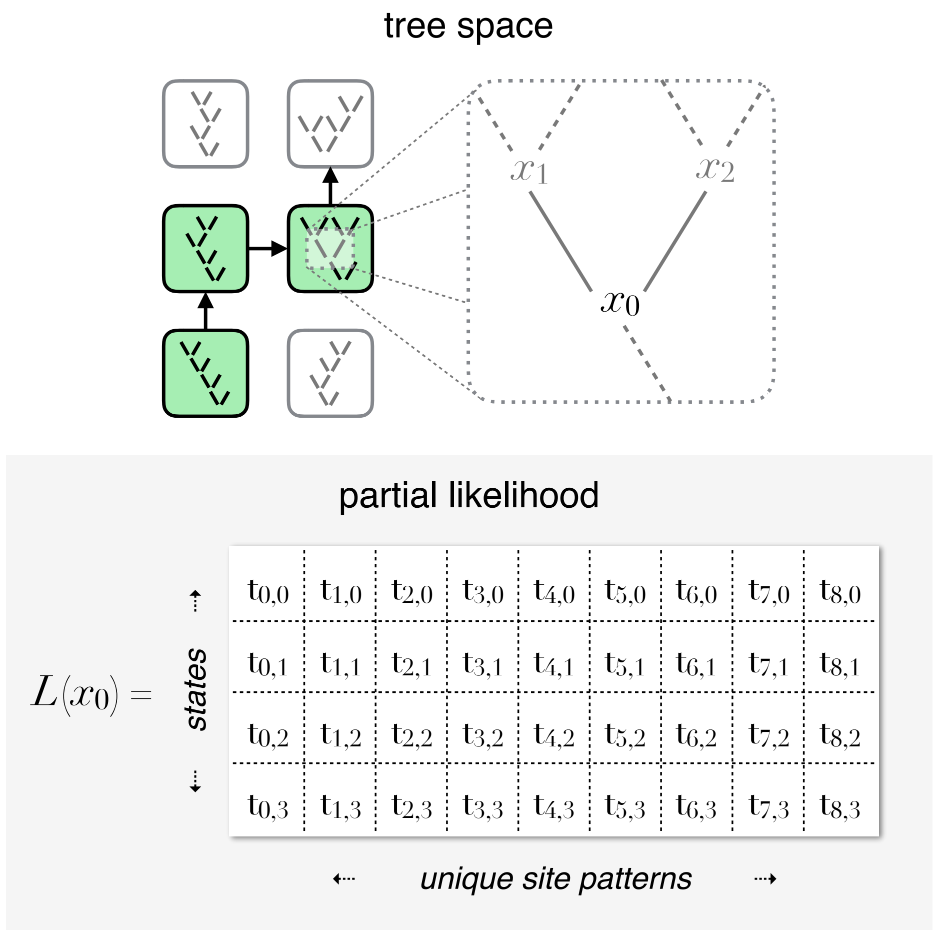

tree space
x_1
x_2
x_0
partial likelihood
$L(x_0) =$
states
$t_{0,0}$ $t_{1,0}$ $t_{2,0}$ $t_{3,0}$ $t_{4,0}$ $t_{5,0}$ $t_{6,0}$ $t_{7,0}$ $t_{8,0}$
$t_{0,1}$ $t_{1,1}$ $t_{2,1}$ $t_{3,1}$ $t_{4,1}$ $t_{5,1}$ $t_{6,1}$ $t_{7,1}$ $t_{8,1}$
$t_{0,2}$ $t_{1,2}$ $t_{2,2}$ $t_{3,2}$ $t_{4,2}$ $t_{5,2}$ $t_{6,2}$ $t_{7,2}$ $t_{8,2}$
$t_{0,3}$ $t_{1,3}$ $t_{2,3}$ $t_{3,3}$ $t_{4,3}$ $t_{5,3}$ $t_{6,3}$ $t_{7,3}$ $t_{8,3}$
unique site patterns

Figure 4.2: Diagrammatic example of the tree sampling process and fine-grained parallel computation of phylogenetic partial likelihoods using BEAGLE on GPUs for a nucleotide-model problem with 5 taxa and 9 site patterns. Each entry in a partial likelihood array L is assigned to a separate GPU thread t. In this simplified example, 36 GPU threads are created to enable parallel evaluation of each entry of the partial likelihood array $L(x_0)$ Calculations for real datasets would typically generate thousands of threads.

4.5 Extending Supported Hardware in BEAGLE

4.5.1 Benchmarking and Testing Methods

As we extended BEAGLE with new implementations, we further developed our test program (*genomictest*) to support a wider range of analysis types and more detailed output. This program generates random synthetic datasets of arbitrary sizes and is used to evaluate performance and assure correct functioning of the library.

For benchmarking we generate a measure of throughput in terms of the effective number of floating point operations per second for computation of the partial-likelihoods function (see equation 4.1 on page 56). In contrast to a direct timing benchmark, throughput allows us to more easily compare performance across different problem sizes and floating point precision formats. This measure also allows comparison to an upper performance bound and generally informs whether computations are compute or memory bound.

For assessing result correctness, we developed a set of testing scripts which evaluate different analyses types by varying input parameters to our *genomictest* program. These testing scripts are publicly available in the project repository and we have verified correct functioning of all new implementations described below.

Table 4.1 on the next page shows relevant hardware and software specifications for the two main systems used to perform the benchmarks results reported in this paper. Table 4.2 on the following page summarizes the hardware features of the

Table 4.1: System Specifications

	system 1	*system 2*
CPU (Intel)	Core i7-930	Xeon E5-2680v4 (x2)
GPU 1 (NVIDIA)	Quadro P5000	—
GPU 2 (AMD)	Radeon R9 Nano	FirePro S9170
Linux kernel	4.8.13	3.10.0
GCC version	6.2.1	6.2.0
CUDA release	8.0	—
OpenCL driver 1	NVIDIA 375.26	Intel 1.2.0
OpenCL driver 2	AMD 1912.5	AMD 1800.8

Table 4.2: GPU Specifications

	Quadro P5000	Radeon R9 Nano	FirePro S9170
Cores	2560	4096	2816
Memory	16 GB	4 GB	32 GB
Bandwidth	288 GB/s	512 GB/s	320 GB/s
SP compute	8900 GFLOPS	8192 GFLOPS	5240 GFLOPS

three GPUs used, with *Bandwidth* denoting device global memory bandwidth and *SP compute* indicating theoretical single-precision peak throughput.

4.5.2 Design Modifications

In order to support additional hardware devices, we have modified the BEAGLE library at different levels (Fig. 4.3 on the next page). At the implementation base-code layer we have changed the serial CPU solution to a *threaded model* one, using C++ threads, and throughout this paper C++ refers implicitly to the 2011 version of the standard [115]. We have also modified what was the CUDA base-code to a framework independent *accelerator model* with support for both CUDA and OpenCL external computing frameworks. This parallel implementation model communicates with the CUDA and OpenCL APIs through a single internal interface, which, in turn, has an implementation available for each framework.

Further significant sharing of code between CUDA and OpenCL exists at the kernel level. There is a single set of kernels for both frameworks, with keywords for each being defined at the pre-processor stage. Though there is a common kernel code-base for both frameworks, functions that impart a crucial effect on performance are differentiated for each hardware type. This allows for distinctly optimized parallel implementations that are shown on the figure, one for CUDA GPUs, one for OpenCL GPUs, and one for parallel x86 devices such as multicore CPUs with SIMD-extensions.

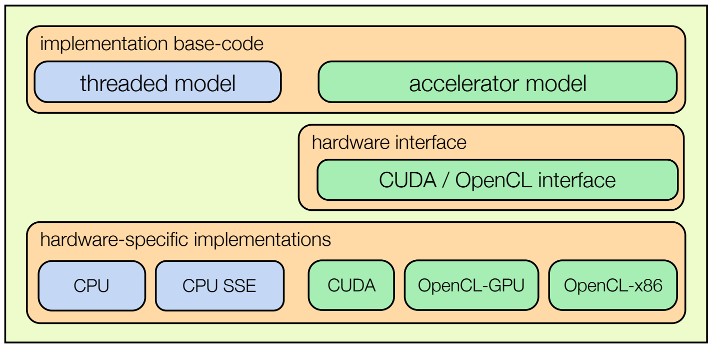


Figure 4.3: Layer diagram depicting the modified portions of the BEAGLE library necessary to extend hardware support.

4.5.3 Library Availability

The BEAGLE project is open source under the GPL v3.0 license. The work described here will be part of an upcoming release and is available under a development branch of the library located at https://github.com/beagle-dev/beagle-lib/

tree/kernel-concurrency.

The library includes compilation workflows for all major platforms (Linux, macOS, Windows). For Linux and macOS it uses an autoconf/automake build system. For Windows we use a Visual Studio build system. One unique aspect of the compilation system is the use of scripts to generate OpenCL/CUDA kernel source code for different inference types (e.g., amino-acid or codon-based) and floating point formats, allowing for better performance at runtime.

4.6 CPU Threaded Implementation

To harness the increasingly parallel nature of modern CPUs, and recognizing that external frameworks such as CUDA and OpenCL are not always available to users of BEAGLE, we developed a more portable parallel implementation.

In the process of developing our solution, we briefly assessed a variety of CPU threading frameworks such as POSIX threads and OpenMP. Ultimately, we felt the best solution when balancing portability, development cost, and performance was to use the C++ threading model. This approach also allowed us to more easily combine the added parallelism with the existing, low-level, SSE vectorization of character states.

Given the decision to use C++ threads, we then iterated through a variety of approaches to concurrent computation of the phylogenetic likelihood function which we will briefly describe and compare below.

4.6.1 Futures

Our initial approach involved modifying the default CPU implementation in BEAGLE such that for each partial-likelihoods operation to be computed, a C++ standard library asynchronous future was created. Thus, this approach only concurrently computed partial-likelihood operations that were independent in the tree topology being assessed, and did not take advantage of the independent nature of each sequence pattern in the likelihood computation.

4.6.2 Thread-create

Our next approach involved the on-demand creation and joining of a set of threads with each partial-likelihoods call to BEAGLE. These C++ standard library threads were used for concurrent computation of the partial-likelihood functions across independent site patterns. We used a load-balancing approach wherein the sequence of independent patterns is broken up into equal sizes, according to the number of CPU hardware threads available. To prevent small problem sizes from being slower than the previous serial implementation, we set a minimum sequence length of 512 patterns for threading to be used.

4.6.3 Thread-pool

This final iteration of our CPU threading solution involved modifying the *thread-create* approach to use a pool of C++ standard library threads. For this approach we also used the threads for concurrent computation of the root likelihood

Table 4.3: CPU threading optimizations

	throughput (GFLOPS)				speedup
tips	*serial*	*futures*	*thread-create*	*thread-pool*	(× serial)
8	35.82	37.92	39.07	**193.10**	5.39
16	35.47	59.70	78.26	**258.99**	7.30
64	14.95	78.67	87.91	**217.24**	14.53
128	13.62	61.61	60.19	**126.95**	9.31

across independent site patterns, in addition to the partial-likelihoods function.

Table 4.3 compares the relative performance of the core partial-likelihoods function for each of the threading approaches we assessed. The throughput measure in GFLOPS is for the single-precision floating point format and is computed as described in Section 4.5.1 on page 63.

For this comparison we used a fixed sequence length of 10,000 patterns across tree sizes of 8, 16, 64, and 128 sequences at the tips, running on the two CPUs on *system 2* (Table 4.1 on page 64). The column labeled *serial* shows throughput for the original single-threaded CPU implementation in BEAGLE, with some degree of vectorization provided by GCC.

The results show the increases in performance for each iteration of our CPU threading solution and that the *thread-pool* approach performs best across all four problem sizes assessed. We also note the relative increase in performance from the original serial implementation to the final *thread-pool* solution.

4.7 OpenCL Implementation

In order to exploit a broader range of hardware resources, including AMD and Intel GPUs, we extended BEAGLE so it can use the OpenCL programming framework, an open standard for parallel computing devices. With OpenCL we have also been able to better utilize the parallel computing capability of modern CPUs, both via multiple cores and vectorization extensions such as Intel SSE and Advanced Vector Extensions (AVX).

4.7.1 OpenCL and CUDA Code Sharing

The OpenCL work is based on our previous implementation for the CUDA platform. Taking advantage of the many similarities between these parallel-computing frameworks we developed a shared code design that includes a single internal interface to the hardware resource and a single set of kernels. This design allows future work on the library to more easily benefit users of either framework.

A single set of kernels for OpenCL and CUDA is achieved by using preprocessor definitions for framework specific keywords. The internal hardware interface is also shared, and only the implementation itself differs between OpenCL and CUDA. The hardware interface deals with loading the different kernels and compiling the correct one for the given analysis parameters (such as the number of states the model can assume, and floating point precision), as well as all the hardware accelerator related functions such as executing kernels, copying data, querying device characteristics, and other auxiliary functions. Few further distinctions had to be overcome for both frameworks to share code. Most notably, subpointer addressing within kernels was done by using the *clCreateSubBuffer* function in OpenCL and by pointer arithmetic in CUDA.

4.7.2 Hardware-Specific Optimizations

The use of OpenCL provides a common, vendor-neutral, platform for current and future parallel hardware architectures and allows BEAGLE to exploit a variety of resources from a single code base. Nonetheless, recognizing that important distinctions exist between what practices work best for each hardware architecture, we have adapted performance-critical code for different runtime scenarios. These hardware-specific optimizations can be categorized into two variants of our OpenCL solution, one that targets GPU architectures and another that addresses x86 processors.

4.7.2.1 OpenCL-GPU

With our OpenCL-GPU solution we focused on high-end NVIDIA and AMD GPUs, though our implementation is also compatible with Intel GPUs.

Our initial work on OpenCL consisted of a direct translation of the CUDA implementation running on the same NVIDIA hardware. Although we expect NVIDIA GPUs to exhibit best performance under CUDA, having them working under OpenCL served as an important comparison point and validation of our approach. The effect of framework-choice on NVIDIA devices is further explored in

Table 4.4: OpenCL-GPU optimizations

		throughput (GFLOPS)		
precision	patterns	without FMA	with FMA	% gain
single	10,000	213.02	**216.87**	1.81
double	10,000	124.14	**136.88**	10.26
single	100,000	408.63	**411.43**	0.69
double	100,000	178.04	**199.23**	11.90

Section 4.8 on page 74.

For AMD GPUs, we found that few changes were required, as these are ultimately similar in architecture to NVIDIA CUDA devices. For codon-based inference models and others with higher-count state spaces, we had to reduce the number of sequence patterns computed per work-group in our likelihood calculation kernel. This was in order to reduce memory usage in the *local* address space, as we found AMD devices to have less of this memory than NVIDIA devices.

Another optimization we implemented for AMD GPUs was the use of the OpenCL precompiler definitions FP_FAST_FMAF and FP_FAST_FMA, for single and double-precision floating-point operations respectively. These macros achieved non-trivial performance gains without loss of precision, and indicate whether fast fused-multiply-add (FMA) operations, which perform multiply and add operations in a single action, are supported by the hardware. For a problem size of 10^5 sequence patterns on a modern GPU (AMD Radeon R9 Nano on *system 1*), we noticed up to an 11.9% performance improvement in double-precision mode for our core partial-likelihoods kernel (Table 4.4).

4.7.2.2 OpenCL-x86

For our OpenCL-x86 solution we collaborated with Intel to develop an optimized implementation for Xeon CPUs and first generation Xeon Phi (Knights Corner) accelerators. Since this work started Intel has dropped support for OpenCL on Xeon Phi, however Intel has continued developing drivers for Xeon CPUs and we have observed strong performance on these multicore processors with our x86 solution (further detailed in Section 4.8 on page 74).

In the process of finding the best solution for Intel x86 processors, we tested several variations of our core partial-likelihoods kernel. This included explicit OpenCL vector usage and reorganization of execution threads from two to three-dimensional work-groups. Ultimately we found that the key optimization was to have each thread of execution do more work in comparison to our GPU approach. This was especially important when computing the partial-likelihoods function for nucleotide models where only 4 states are possible and each thread has a lighter workload. To achieve this heavier workload per thread, our OpenCL-x86 for DNA-based inferences, loops over the state space in each work-item instead of computing all states concurrently, as is done with the GPU approach. We also found that it was advantageous to avoid the explicit use of the *local* memory address space and allow the OpenCL compiler to manage memory caching.

Given these x86-specific changes to our nucleotide-model likelihood computation kernel, we proceeded to optimize for work-group size, which determines the number of sequence patterns computed per work-group. Table 4.5 explores perfor-

Table 4.5: OpenCL-x86 optimizations

solution	work-group size (patterns)	throughput (GFLOPS)	speedup (× OpenCL-GPU)
OpenCL-GPU	64	15.75	
OpenCL-x86	64	79.65	5.06
	128	85.51	5.43
	256	**98.36**	**6.25**
	512	98.09	6.23
	1024	96.51	6.13

mance with the dual CPUs on *system 2* for work-groups of increasing size. The table also shows throughput for our original OpenCL-GPU solution running on the Xeon CPUs and the relative speedup achieved due to our architecture-specific optimizations. We observe that peak performance is achieved with a work-group size of at least 256 patterns. We opted to use this size as we prefer the smallest work-group size with peak or near-peak performance to reduce pattern padding when the total number of patterns is not divisible by the work-group size.

4.7.2.3 OpenCL Driver Implementations

BEAGLE makes use of the OpenCL Installable Client Driver loader to make all implementations on a system available, which allows the selection of different drivers for the same hardware resource.

On Linux and Windows operating systems we have found that vendor-specific OpenCL driver implementations offer the best performance. On macOS vendor-specific drivers are not available and we observed reduced performance compared to other platforms.

4.8 Results

Here we explore the performance of the new implementations for the BEAGLE library on a variety of modern parallel hardware resources. System specifications are as described in Table 4.1 on page 64. We also evaluated performance on a machine with an Intel Xeon Phi 7210 CPU (not an accelerator), Linux kernel version 3.10.0, and GCC version 6.2.0.

4.8.1 Partial-likelihoods Kernel Performance

We have used our *genomictest* program to benchmark the core likelihood function of BEAGLE on a variety of hardware platforms and for a range of problem sizes. Again, this function is the main bottleneck for phylogenetic inferences, typically accounting for over 90% of the total execution time. We have found the relative performance gains observed here correlate strongly with those of a full inference run.

Figure 4.4 on the next page shows throughput in effective GFLOPS (billions of floating-point operations per second) for our partial- likelihoods calculation kernel for analyses with increasing unique site pattern counts, across a number of parallel computing devices and implementations. We evaluated our C++ threading, OpenCL-x86, OpenCL-GPU, and CUDA implementations. The hardware devices represent a sample of the range of consumer-level and high-performance computing resources available to domain scientists who are the ultimate users of the BEAGLE library, and included AMD Radeon R9 Nano, AMD FirePro S9170, and NVIDIA

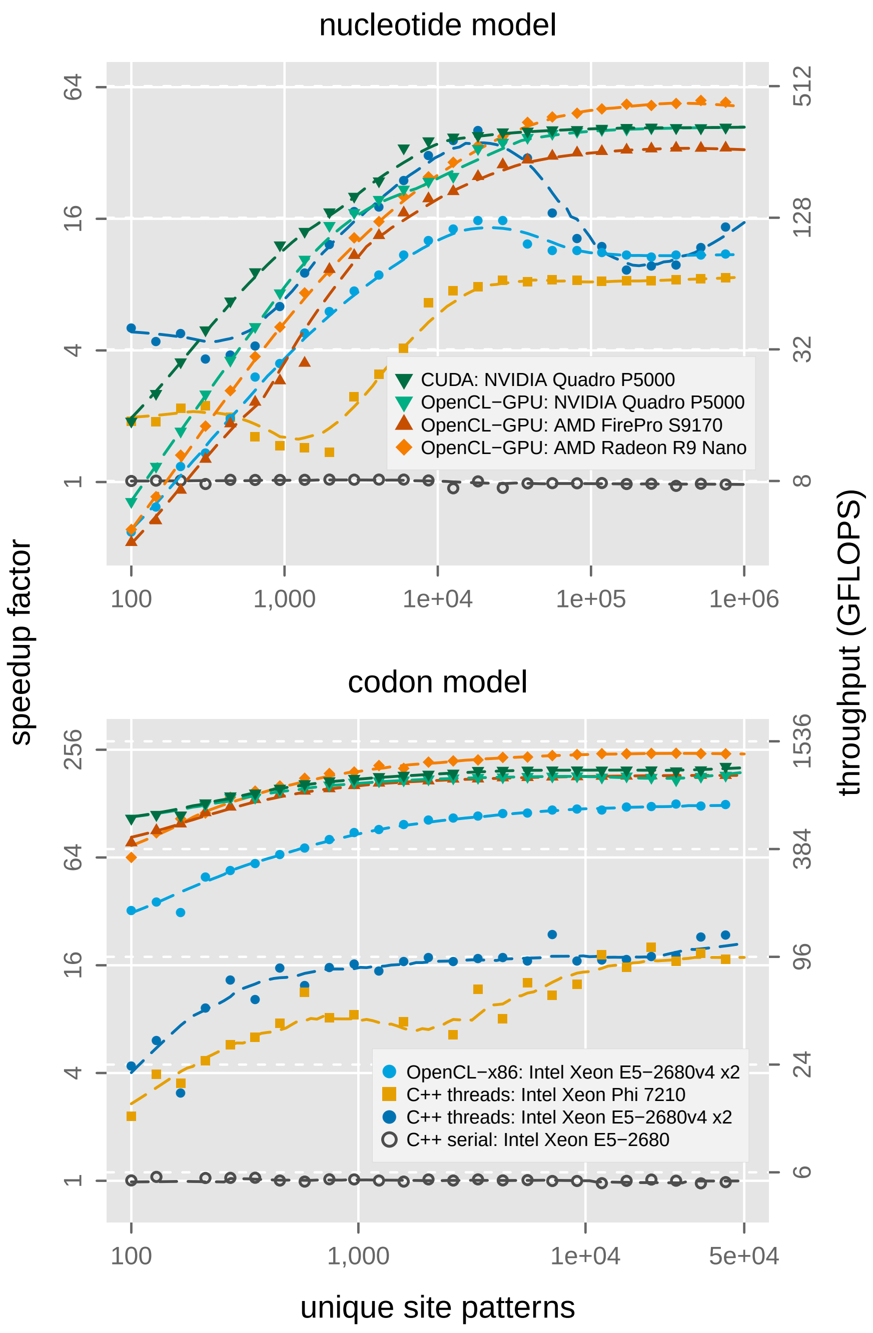


Figure 4.4: Plots showing throughput performance in GFLOPS for the core likelihood function of the BEAGLE library, for nucleotide and codon-based models with a range of problem sizes running on a variety hardware platforms and implementations. Speedup factors (which are relative to unvectorized single-core performance), throughput, and number of unique site patterns are on a log-scale.

Quadro P5000 GPUs, Intel Xeon Phi 7210 manycore CPU, and dual Intel Xeon E5-2680v4 multicore CPUs. The figure includes performance results for computing partial-likelihoods for both nucleotide and codon-model analyses. The left-side vertical axis labels show the speedup relative to the average performance of a baseline serial, single threaded and non-vectorized, CPU implementation. We chose to use this non-parallel CPU implementation as a comparison baseline as it provides a consistent performance level across different problem sizes. It is also relevant as it has been the default implementation in BEAGLE in previous releases and many phylogenetic inference softwares use serial code as their standard. We note that performance results reported here are in all cases far from theoretical peak compute throughput for each platform as the calculation of phylogenetic sequence likelihoods is substantially memory-bound, especially for nucleotide models.

4.8.1.1 Nucleotide Model

For nucleotide-based likelihood, we observe that throughput strongly scales with the number of site patterns for all parallel hardware resources using our *accelerator model*. For a small number of patterns the parallel OpenCL implementations exhibit poor performance relative to others due to greater execution overhead. By 10^5 patterns the performance across these devices has reached a saturation point, with the exception of the AMD Radeon R9 Nano GPU, which continues to slightly scale up in performance. Overall, best performance is achieved by the AMD Radeon R9 Nano GPU, with 444.92 GFLOPS of throughput for a problem with 475,081 unique site patterns. This represents a ∼58-fold speedup over the baseline serial, non-vectorized, CPU implementation, and a ∼5.1-fold speedup over our fastest CPU solution at this problem size, which is the OpenCL-x86 implementation running on two Intel Xeon E5-2680v4 processors.

For CPUs using our *threaded model*, we observe that performance does not monotonically increase with the number of patterns and we require further investigation to understand this aspect of the result. We observe very strong performance for the dual Intel Xeon E5-2680v4 CPUs between approximately 3,000 and 50,000 patterns, with a peak performance of 328.78 GFLOPS at 20,092 unique patterns, also being the overall fastest implementation at this problem size. We observe weak performance from the Xeon Phi 7210 CPU for problems under 10^4 patterns, though we have not done optimization work specific to this platform. Further, we note that we did not use SSE vectorization for these benchmarks as it is not available in single-precision in BEAGLE.

4.8.1.2 Codon Model

For codon-based analyses, we observe that throughput performance is less sensitive to the number of unique site patterns. This is due to the better parallelization opportunity afforded by the 61 biologically-meaningful states that can be encoded by a codon. This higher state count of codon data compared to nucleotide data increases the ratio of computation to data transfer resulting in increased relative performance for codon-based analyses (Fig. 4.4 on page 75). We also observe similar performance from all GPU devices and less overhead effect from use of the OpenCL framework. Overall, highest throughput is achieved by the AMD Radeon R9 Nano GPU, with 1324.19 GFLOPS for 28,419 patterns, equivalent to a $\sim$253-fold speedup over the baseline serial, non-vectorized, CPU implementation and $\sim$2-fold speedup over the OpenCL-x86 implementation running on two Intel Xeon E5-2680v4 processors. Our *threaded model* for CPUs does not perform as well for codon-based inferences as it only parallelizes the computation of independent site patterns.

4.8.2 Multicore Performance Scaling

Figure 4.5 on the next page shows CPU performance results of the core likelihood function for nucleotide-model analyses with 10^4 unique patterns when utilizing an increasing number of hardware threads on *system 2*. The two processors on this system have 14-cores each for a total of 56 hardware threads running at 2.40 GHz. This benchmark was achieved using the *taskset* utility in Linux for our *threaded model* implementation, and the OpenCL device-fission feature for our OpenCL-x86 solution.

Parallelization on multicore systems remains an important topic as researchers increasingly invest in multicore hardware, where core counts on high-end systems regularly reach 40 or greater. The results here show that throughput for both implementations starts to saturate at around 27 threads, suggesting memory bandwidth limitations.

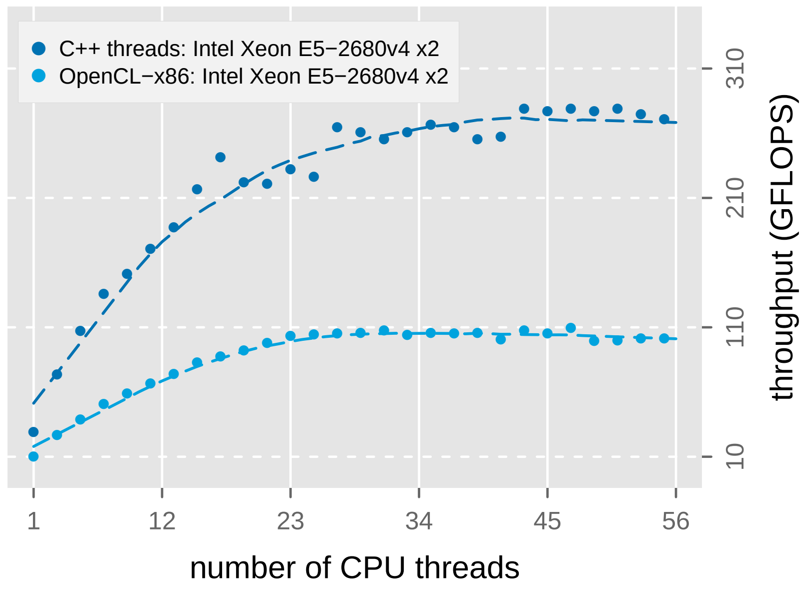


Figure 4.5: Plot showing multicore CPU performance scaling for the *threaded model* and OpenCL-x86 implementations in BEAGLE for nucleotide-model likelihood function with 10^4 patterns. Throughput in GFLOPS is on a log-scale.

4.8.3 Application-Level Results

We ran MrBayes 3.2.6 on *system 2* to benchmark application-level performance for our new C++ threaded, OpenCL-x86, and OpenCL-GPU implementations for BEAGLE. MrBayes uses MPI to concurrently compute separate Markov chain Monte Carlo chains across processors [28]. This is an additional level of concurrency and is complimentary to that provided by the BEAGLE library, which parallelizes computation across site patterns with our *threaded model* implementation, and across site patterns, states, and rate categories with our *accelerator model* solutions. Additionally, MrBayes uses SSE vectorization in single-precision floating point format.

For evaluating performance with the nucleotide model we used a dataset from an RNA-Seq study of advanced moths and butterflies [116] with 16 taxa and 742,668 site patterns, of which 306,780 were unique. For the codon model benchmark we used a 15 taxa dataset with 6,080 unique codon patterns, which was a subset of a larger arthropod dataset [80]. Both analyses were run with four Metropolis-coupled, Markov chain Monte Carlo chains.

We also assessed each dataset under single and double-precision floating point formats. MrBayes supports both modes and certain analyses with larger number of taxa benefit from more precise computation. All reported speedups compare the total execution time relative to that of MrBayes-MPI in double-precision mode.

Generally we observe that speedups are largest under the codon models, as they allow for greater parallelism. For the OpenCL-GPU implementation we note

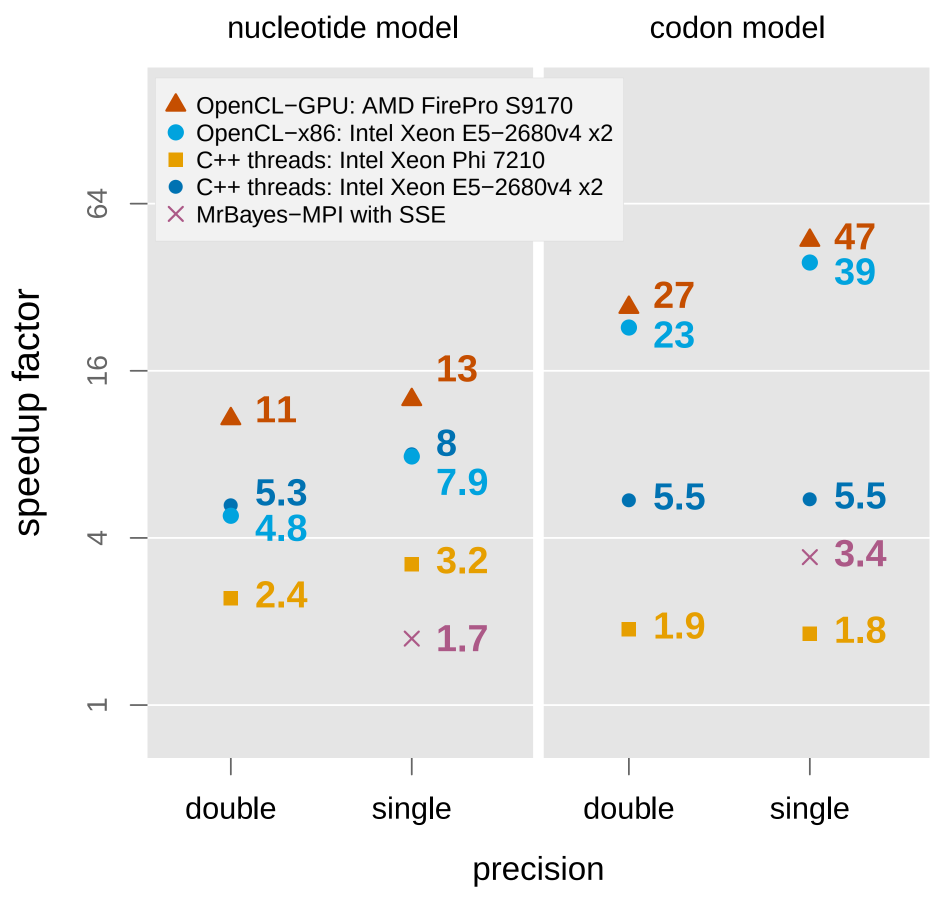


Figure 4.6: Plot showing speedups for MrBayes in single and double-precision mode using various BEAGLE library implementations as well as the built-in SSE option, relative to the MrBayes-MPI implementation in double-precision. Speedup factors are on a log-scale.

significant speedups across all benchmark scenarios. Relative to the fastest single-precision format implementation in MrBayes, speedups are 7.6 and 13.8-fold for the nucleotide and codon model analyses, respectively. For the CPU-based implementations, we observe that for a nucleotide analysis of this problem size both implementations are closely matched, although for codon inferences, the OpenCL-x86 has a significant advantage. We also observe relatively modest performance from the Xeon Phi CPU across all scenarios.

4.9 Conclusion

The BEAGLE project addresses a common bottleneck across phylogenetic inference programs by accelerating likelihood computation. The library now includes additional parallel computing implementations, and combines both CUDA and OpenCL frameworks in a single codebase to address a wider-range of hardware resources. These advancements are of immediate benefit to users of phylogenetic programs that exploit the library. Additionally, developers of other phylogenetic software packages can reference these results to assess the suitability of using BEAGLE with their program, or for developing similar parallel solutions.

Although the improvements described in this paper also allow users to execute in parallel on multiple devices within a system, this requires the client program to partition the problem across site patterns and create a separate library instance for each hardware device. Further, selecting the best performing implementation depends not only on the hardware available but on problem size and type. We plan to further develop BEAGLE so that computation can be dynamically load balanced across multiple devices from within a single library instance. The library would also select the best implementation for each data subset and hardware pair. This will allow for greater memory efficiency and performance gains which will be especially relevant in heterogeneous systems.

Acknowledgment

We thank Marc Suchard, University of California, Los Angeles, and Andrew Rambaut, University of Edinburgh; Yariv Aridor and Arik Narkis, Intel Israel, for assistance with Intel Xeon Phi programming; Mark Berger of NVIDIA; and Greg Stoner and Ben Sander of AMD. This work was supported by the National Science Foundation grant numbers DBI-0755048 and DBI-1356562.

Chapter 5: Configuring Concurrent Computation of Phylogenetic Partial Likelihoods: Accelerating Analyses using the BEAGLE Library

5.1 Introduction

The most effective methods for inferring phylogenetic trees are based on either maximum likelihood estimation or Bayesian analysis, which share the same computational bottleneck: calculation of the likelihood of trees [6]. When profiling GARLI [2], a leading phylogenetic inference program, we have observed that, for nucleotide models, likelihood related calculations typically constitute over 94% of the overall run time. For more complex models (e.g., amino-acid or codon-based), likelihood calculation will typically incur an even greater proportion of the analysis time. Speeding the calculation of the likelihood function is key to increasing the performance of statistical inference-based phylogenetic analyses.

The core likelihood calculations apply to a subtree comprising a parent node, k, two child nodes, ℓ and m, and connecting branches of length, t_ℓ and t_m, and is repeated for all such subtrees within the larger tree being considered. This partial likelihood function is as follows [6]:

$$L_k^{(i)}(z) = \left(\sum_x \Pr(x|z, t_\ell) L_\ell^{(i)}(x) \right) \times \left(\sum_y \Pr(y|z, t_m) L_m^{(i)}(y) \right) \tag{5.1}$$

This calculation is repeated for each character i in the data (i.e., sequence site pattern), for each state z that a character can assume, and for each internal node in the proposed tree. The computational complexity of the likelihood calculation for a given tree is $O(p \times s^2 \times n)$, where p is the number of patterns in the sequence (typically on the order of 10^2 to 10^6), s is the number of states each character in the sequence can assume (typically 4 for a nucleotide model, 20 for an amino-acid model, or 61 for a codon model), and n is the number of operational taxonomic units (e.g., species, alleles). Additionally the tree search space is very large; the number of unrooted topologies possible for n operational taxonomic units is given by the double factorial function $(2n - 5)!!$ [7]. Thus, to explore even a fraction of the total search space, a very large number of topologies are evaluated, and hence a very great number of likelihood calculations have to be performed. This leads to analyses that can take days, weeks or even months to run. Further compounding the issue, rapid advances in the collection of DNA sequence data have made the limitation for biological understanding of these data an increasingly computational problem.

5.1.1 The BEAGLE Library and API

The BEAGLE library and API [12] is a high-performance likelihood-calculation platform for evolutionary models. It defines a uniform application programming interface (API) and includes a collection of efficient implementations for calculating a variety of likelihood-based models on different hardware devices, such as graphics processing units (GPUs) and multicore central processing units (CPUs).

The BEAGLE library was designed to support a variety of hardware-specific implementations, each optimized for a different processor type. The library includes a set of parallel computing implementations that use the CUDA and OpenCL external computing frameworks.

The BEAGLE library has been very successful in accelerating evolutionary analyses. The library has been integrated into the most recent versions of popular phylogenetics software including BEAST [5], MrBayes [4], and PhyML [37], and has been widely used across a diverse range of evolutionary studies.

Previously, given the fine-scale parallelization of the phylogenetic likelihood function in the BEAGLE library, the problem with few sequence patterns, or one broken into small data subsets, was always *small*, and thus generally not amenable to speedups, as patterns (for a given model type and category rate count, e.g., nucleotide with four distinct rates) were the only dimension being parallelized.

In this paper we describe our recent work to configure concurrent computation of phylogenetic likelihoods by exploiting additional independent calculation opportunities. The result is that a wider variety of analyses benefit from parallel computing performance gains.

5.1.2 Concurrent Computation: Independent Likelihood Estimates

We have focused on the following opportunities for concurrent computation of phylogenetic likelihoods that were previously unrealized in BEAGLE.

5.1.2.1 Pattern Partitions

Evolutionary analyses benefit from increases in modeling flexibility. One clear way of improving model flexibility is to allow independent estimation of model parameters for different character data subsets (e.g., genes, codon positions). This is typically referred to as a partitioned model and is a technique available in all phylogenetic software packages that support BEAGLE. Until now partitioned analyses with BEAGLE have required the client program to create multiple instances of the library, one for each data subset defined by the partitioning scheme. When BEAGLE instances share a hardware resource they are executed in sequence, thus incurring significant performance and memory inefficiencies, specially for problems with a large number of small data subsets.

5.1.2.2 Independent Subtrees

The number of subtrees requiring calculation for any full tree is $n - 1$, where n is the number of operational taxonomic units (e.g., species, alleles), which is the number of tips (leaves) on the tree. Phylogenetic algorithms typically use a post-

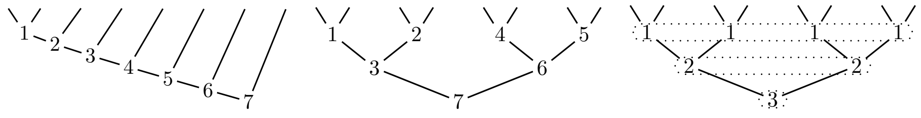


Figure 5.1: Example pectinate tree (*left*), and example of a fully balanced tree (*middle*); with sequential calculation both trees require $n - 1 = 7$ partial likelihood operations in series, corresponding to the order of the node numbers. Balanced tree (*right*) with concurrent computation requiring $\lceil log_2 n \rceil = 3$ sets of independent partial likelihood operations in the order of the shared node numbers.

order traversal when calculating tree likelihood, calculating each of the $n-1$ subtrees in series. In the case of a fully pectinate tree no subtrees are independent (Fig. 5.1, left). However, in the case of more balanced topologies there are independent subtrees (Fig. 5.1, middle). The likelihoods for sets of these independent subtrees can be calculated concurrently. In order to more easily realize potential concurrency related to independent subtrees present in a given topology, partial likelihood arrays need to be processed according to a reverse level-order, or breadth-first, traversal of the tree being evaluated. In the case of a fully balanced tree the number of independent subtrees is maximized, and partial likelihood calculations can be done in sets of concurrent operations corresponding to the number of levels in the tree, $\lceil log_2 n \rceil$ (Fig. 5.1, right). This exploit of tree level-group concurrency is somewhat similar to a classic parallel reduction scheme.

5.2 Methods

5.2.1 Benchmarking and Testing

Our approach to increase concurrency in BEAGLE has been focused on the partial likelihoods kernel that is the computational bottleneck for phylogenetic analyses. To evaluate the performance of this function we used our test program (*genomictest*), which generates random synthetic datasets of arbitrary sizes. This test program is included with the BEAGLE source code and the results shown throughout this paper can be reproduced by using the default random seed, 1.

We report a measure of throughput in terms of the effective number of floating point operations per second (GFLOPS) for computation of the partial likelihoods function (see equation 5.1 on page 85). In contrast to a direct timing benchmark, throughput allows us to more easily compare performance across different problem sizes. We report benchmark results for two system configurations (Table 5.1 on the following page). For conciseness, many results are shown only for the two best performing platforms we had available, the NVIDIA Quadro P5000 GPU under CUDA and the AMD Radeon R9 Nano GPU under OpenCL. Further comparisons across hardware platforms and frameworks are reported elsewhere [35].

Table 5.1: System specifications

	system 1	*system 2*
CPU(s)	Intel Core i7-930	dual Intel Xeon E5-2680v4
GPU(s)	AMD Radeon R9 Nano NVIDIA Quadro P5000	AMD FirePro S9170
Linux kernel	4.8.13	3.10.0
GCC version	6.2.1	6.2.0
CUDA release	8.0	—
OpenCL drivers	AMD 1912.5 NVIDIA 375.26	AMD 1800.8 Intel 1.2.0

5.2.2 Pattern Partition Concurrency

5.2.2.1 Multiple versus Single Library Instances

An initial design goal for the BEAGLE library was to make a library instance relatively light-weight, and to leave it up to the client program to manage these instances. This design objective was fitting for processors at the time, because it was easier to achieve good saturation as the number of cores and supported threads for CPUs and GPUs were modest compared to recent processors. However, we have found that this light-weight model is limited, as the client program does not have direct access to the parallel devices and cannot configure concurrent communication efficiently. Furthermore, this model of separate instances also limits us to the concurrency afforded to asynchronous kernel executions by the parallel computing framework used (i.e., CUDA or OpenCL).

Given our desire to improve concurrency for partitioned analyses, our first decision was to move away from one library instance per data subset. This gave us greater potential for concurrency, such as via single kernel launches, and more control over how computation is combined into concurrent executions. Using a single library instance also results in significant memory savings given many overhead costs become shared for all partitions.

5.2.2.2 API Changes

In order to support partitioning in a single library instance we have modified the BEAGLE API to support data subset assignment and per-subset operations. Partition assignment can be done via a pattern-count length array of integers, with support for noncontiguous assignments. These changes were done as additions to the existing BEAGLE v1 API, and the interface remains backwards compatible.

5.2.2.3 CUDA First

Our work to increase concurrency, and thus efficiency, for partitioned analyses initially focused on our parallel implementation for the CUDA framework. We have found this framework to be generally more mature than OpenCL, and to support more features. We identified two solutions to allow independent data subsets to be concurrently computed: a) using CUDA *streams*, which would allow separate likelihood kernel launches to run concurrently; and b) developing a *multi-operation* likelihood kernel, which would compute multiple likelihood arrays within a single kernel launch. Below we describe each approach.

5.2.2.4 Streams

This feature of the CUDA framework is described by NVIDIA as follows:

> "The CUDA programming model provides streams as a mechanism for programs to indicate dependence and independence among kernel launches. Kernels launched into the same stream are guaranteed to execute consecutively, while kernels launched into different streams are permitted to execute concurrently. Streams describe independence between work items and hence allow potentially greater efficiency through concurrency."

To achieve partition concurrency we launch our likelihood kernels on separate streams according to the data subset of the likelihood array operation. We do so in a breadth-first manner, that is, the kernel launch for the first partial likelihood array operation for data subset 1 is followed by the launch for the first operation for subset 2, and so on. This is to compensate for signal delay in each stream. We use this multi-stream approach for both partial likelihood and likelihood integration kernels. For all other kernel launches in BEAGLE we use the `null` stream which synchronizes with all streams.

5.2.2.5 Multi-Operation Kernel

Our second solution for data subset concurrency involved modifying our partial likelihood CUDA kernel to compute multiple likelihood arrays in a single execution launch. We used pointer arithmetic to allow different input and output arrays for different execution blocks.

Figure 5.2 contrasts available data arrays (`nodes`, `branches`) and likelihood array index (`pattern`) for our single and multi-operation partial likelihood kernels. The first implementation is restricted to a single set of input likelihood arrays (for nodes c_1 and c_2), input branch length arrays (t_1 and t_2), and output array (d_0), for all execution blocks. Additionally the pattern computed by each execution thread is directly determined by block index n, block size $blockSize$, and thread index $threadId$.

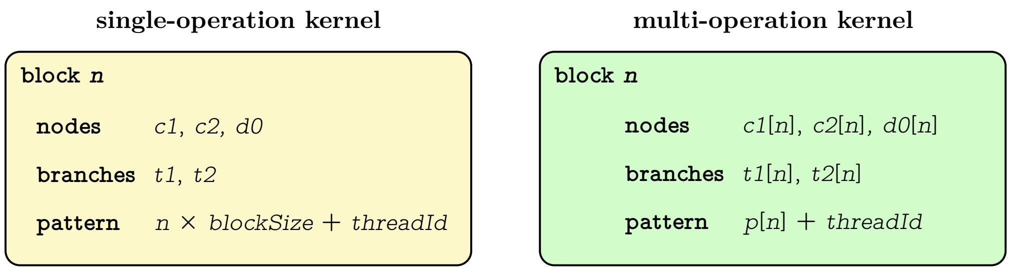


Figure 5.2: Organization of data arrays and indexing for single and multi-operation kernel execution blocks for partial likelihoods computation in BEAGLE.

With the *multi-operation* approach, input and output arrays are determined based on the block index. Further, the pattern computed by each thread is only indirectly determined by n, which allows padding of data subsets when these do not fall along block-sized boundaries.

Additionally, to maximize device global memory throughput we rearrange site patterns on device memory so that data subsets are contiguous. This is done when sequence partition assignment is made by the client program and enables each execution block to operate on a single data subset more efficiently.

5.2.3 Independent Subtree Concurrency

As we developed the above approaches to partition concurrency, we noted we could also leverage those methods to concurrently compute partial likelihood arrays for independent subtrees. This would be specially beneficial for large trees with short sequences when running on manycore processors such as GPUs. This combination of problem size and hardware resource previously left many processing cores underutilized. Below we describe implementation details for independent subtree operations via both our *streams* and *multi-operation* solutions.

5.2.3.1 Streams

We further leveraged the use of CUDA streams to concurrently compute partial likelihood arrays of independent subtrees by assigning them as described by Algorithm 1 on the following page. This algorithm shows how we assign a likelihood array kernel launch (`pLikelihoods`) to a stream based on an inherited index from either of the child nodes (`child1` or `child2`). Additionally, we may wait on a CUDA event that has been recorded for the other child node before launching the kernel.

Algorithm 1: Streams and partial likelihood array operations

```
Data: a sequence of likelihood operations in reverse level-order traversal
Result: computation of partial likelihood arrays in concurrent streams

streamIndex ← 0
foreach operation in the operations sequence do
    node ← operation.parent
    if node.child1.streamIndex is not null then
        node.streamIndex ← node.child1.streamIndex
        node.waitIndex ← node.child2.streamIndex
    else if node.child2.streamIndex is not null then
        node.streamIndex ← node.child2.streamIndex
        node.waitIndex ← node.child1.streamIndex
    else
        node.streamIndex ← streamIndex + 1
        streamIndex ← streamIndex + 1
    end

    if node.waitIndex is not null then
        cudaStreamWaitEvent(event node.waitIndex, stream
         node.streamIndex)
    end
    cudaLaunchKernel(kernel pLikelihoods, stream node.streamIndex)
    cudaEventRecord(event node.streamIndex, stream node.streamIndex)
end
```

5.2.3.2 Multi-Operation Kernel

To implement subtree concurrency with this kernel, we process partial likelihood subtree operations according to a reverse level-order traversal of the proposed tree. We add each consecutive operation to a set until we find an operation that is dependent on the result of a previous operation in the set. We then start a new operation set, repeating the same process. Once we have processed all operations in this manner, we successively launch each operation set for concurrent computation using our *multi-operation* partial likelihoods kernel.

5.2.4 Extending Concurrency Gains to OpenCL

Our next step was to extend the above work, using the CUDA framework, to our OpenCL implementation.

5.2.4.1 Queues

The OpenCL equivalent to CUDA streams are concurrent execution queues. We implemented our approach in an analogous manner but found the use of concurrent queues only offered at best minimal gains in performance for the OpenCL devices we had access to (AMD Radeon R9 Nano and FirePro S9170 GPUs, and Intel Xeon E5-2680v4 CPU).

5.2.4.2 Multi-Operation Kernel

For this approach, in a comparable manner to CUDA blocks, we launch OpenCL work-groups such that multiple partial likelihood operations can be performed concurrently. In contrast to CUDA, we found that the OpenCL solution was generally more performance sensitive to implementation details such as operation order and synchronization points. This was ultimately beneficial, as we iteratively refined of our likelihood kernel to optimize performance, and could then translate back some of the gains to the CUDA solution.

5.2.5 Memory Transfer Optimizations

For the *multi-operation* approach under either CUDA or OpenCL, we necessitate an explicit memory transfer from host to device for each tree likelihood estimation. Such memory transfers can be costly for GPU devices as they may have to go over the PCI bus. BEAGLE was designed to minimize this type of transfer and previously explicit host to device transfers only occurred at the initialization phase of an inference run.

This additional memory transfer for our *multi-operation* kernel is used to copy the address offsets for the input and output arrays each block in device memory will operate on. In order to minimize costs for this additional memory transfer, we process all subtree operations in a partial likelihoods call to the library, and perform a single transfer for multiple launches of our *multi-operation* kernel.

Further, we use faster methods than we had done before for host to device

Table 5.2: GPU memory transfer optimizations; throughput in GFLOPS

framework	GPU	solution	*tree A*	*tree B*
CUDA	NVIDIA P5000	*write*	328.27	188.76
		pinned	**328.57**	**203.47**
OpenCL	NVIDIA P5000	*write*	320.10	183.78
		map/unmap	**321.24**	**199.58**
	AMD R9 Nano	*write*	397.92	178.04
		map/unmap	**403.72**	**210.30**

transfer: *pinned* host memory allocations under CUDA; and *map* and *unmap* approach with OpenCL. Table 5.2 shows kernel throughput performance with these approaches when compared to the performance when using the regular memory *write* transfer method under each framework. This comparison was done for two tree sizes: *tree A* has 16 tips and 100,032 sequence patterns; and *tree B* has 256 tips and 1024 patterns. We observe that the *pinned* and *map/unmap* approaches have a positive impact on overall throughput, especially for *tree B*, which has many more tips, and thus more partial likelihood operations with an ensuing larger data transfer size.

5.2.6 Combining Pattern Partition and Independent Subtree Concurrency

We have found that the most efficient approach (i.e., *stream/queues*, or *multi-operation*) to concurrent partial likelihood array operations depends on the number of patterns being processed per operation. In order to determine which approach

Table 5.3: Concurrency solutions and partition sizes; throughput in GFLOPS with bold text indicating which concurrency approach within a parallel solution offers best performance at each problem size.

partition		CUDA		OpenCL–GPU		OpenCL–X86	
count	size	*streams*	*multi-op*	*queues*	*multi-op*	*queues*	*multi-op*
1	100,032	**321.82**	272.61	**346.26**	335.62	**79.97**	79.43
2	50,016	**330.08**	228.21	**354.79**	341.02	**79.85**	77.85
16	6,252	**316.72**	225.64	226.77	**330.68**	70.60	**76.10**
24	4,168	**227.63**	223.40	182.71	**318.97**	65.92	**75.21**
32	3,126	164.06	**217.59**	141.50	**317.28**	54.65	**73.00**
64	1,563	87.75	**212.71**	87.98	**326.49**	24.92	**73.61**

to use for different problem sizes, we have benchmarked the throughput for our partial likelihood kernel when evaluating a tree with 16 tips and 100,032 patterns for an increasing number of equal-sized data subsets (Table 5.3) across our different parallel solutions. The CUDA implementation was tested on an NVIDIA Quadro P5000 GPU, the OpenCL-GPU implementation on an AMD Radeon R9 Nano, and the OpenCL-X86 implementation on dual Intel Xeon E5-2680v4 CPUs. Systems were as specified in Table 4.1 on page 64.

With the CUDA implementation, we observe that for larger numbers of patterns (above 4,168) the Quadro P5000 GPU is near saturation, and the one-time overhead of the *multi-operation* approach makes it relatively inefficient (Table 5.3). However, for smaller problem sizes there is less work per stream, and the overhead cost for each stream makes that approach the less efficient alternative. For the OpenCL implementations we observe that the *multi-operation* approach is the most efficient or close to most efficient for any partitioned problem.

Based on these findings, and on further intermediate analyses not shown in Table 5.3 on the previous page, we set a fixed crossover point for each solution which determines which approach is used. For the CUDA implementation we have set this at 4,168 patterns, for the OpenCL-GPU it is set at 8,192 patterns, and for the OpenCL-x86 implementation the *multi-operation* approach is always used. Additionally, client programs can also explicitly request either the *streams* or *multi-operation* implementation via the library API.

5.2.7 Other Aspects

Although BEAGLE supports inferences with models of arbitrary state counts, the work described here has thus far only been implemented for nucleotide model inferences.

It is also worth mentioning that our implementation allows partitions to be reassigned at any point. With each new partition assignment we rearrange patterns in device memory to maintain efficient throughput. This functionality may be used by client programs in the future to enable efficient inference of partition assignments in conjunction with currently inferred parameters.

Finally, we use the `--default-stream per-thread` NVIDIA CUDA compiler (NVCC) option so that each BEAGLE instance runs on a separate default stream. This allows further concurrency gains for other independent work in addition to partitioning, such as Metropolis-coupled, Markov chain Monte Carlo chains or run replicates.

5.2.8 Modifications to MrBayes

In order to fully evaluate the efficacy of the concurrency improvements to the library, we have adapted MrBayes version 3.2.6 to use the new BEAGLE API partitioning extensions. This enabled MrBayes to use a single BEAGLE library instance for computing the likelihood of multiple data subsets. This modified version of MrBbayes is open-source under GPL version 3.0, and is available at `https://github.com/ayresdl/mrbayes-beagle3`.

5.2.9 Library Availability

The BEAGLE project is open source under the GPL v3.0 license. The work described here will be part of an upcoming release, and is available under a development branch of the library located at `https://github.com/beagle-dev/beagle-lib/tree/kernel-concurrency`.

5.3 Results

Here we explore the performance effect of the concurrency gains on various parallel hardware resources. System specifications are as shown in Table 4.1 on page 64.

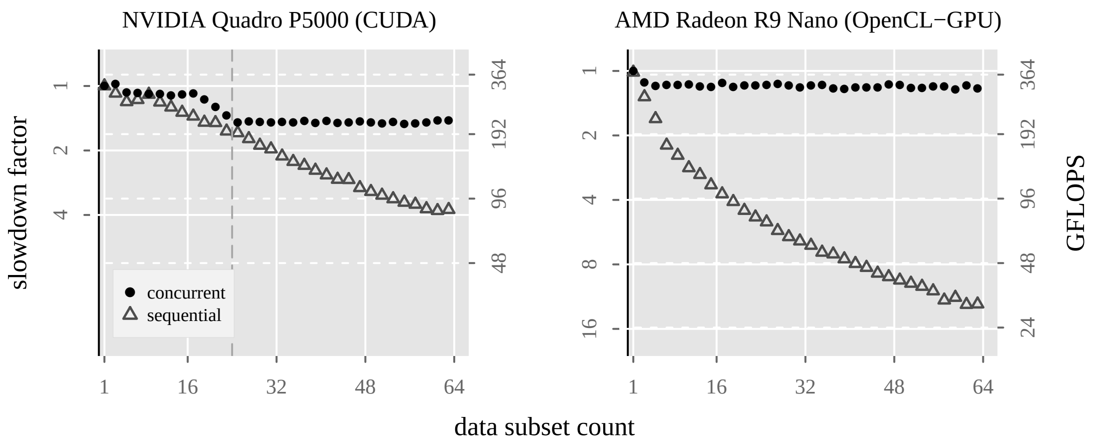


Figure 5.3: Plots showing throughput for the partial likelihood kernel with data subset concurrency (black dots) and with no data subset concurrency (open triangles) for a problem with 100,032 total sequence patterns and increasing number of equal-sized data subsets for two GPU device/framework pairs. Left-axis *slowdown factor* indicates performance loss relative to the unpartitioned case. Slowdown factors and throughput in GFLOPS are on a log-scale.

5.3.1 Pattern Partition Concurrency Gains

We observe that for both the Quadro P5000 and Radeon R9 Nano GPUs the previous approach of sequential computation of data subsets produces a sharp drop-off in throughput as we increase the number of subsets (Fig. 5.3). This is because as we increase the partition count the data subsets have decreasing numbers of patterns, resulting in increasingly underutilized GPU capacity.

For concurrent computation with the CUDA device, throughput is higher than with the sequential approach at all subset sizes. When there are fewer than 24 subsets we use the *streams* approach. Throughput with this approach starts to drop quickly after 17 subsets (corresponding to a subset size of approximately 6,000 patterns). We then note the crossover point at 24 subsets (subset size of 4,168 patterns, and indicated by a dark grey dashed line) where we switch to our *multi-operation* kernel approach. This approach exhibits consistent throughput independent of subset size.

With the OpenCL solution we use the *multi-operation* approach for all partitioned cases and note consistent and near best-case throughput, independent of the number of data subsets.

5.3.2 Independent Subtree Concurrency Gains

Figure 5.4 on the following page shows the performance improvement associated with concurrent computation of independent subtrees for a problem with 512 patterns. The pectinate case (open triangle) also represents performance for any tree topology with our previous solution of serial computation of subtree partial likelihood arrays.

For both GPUs, we observe increasing speedups with tree size for the average random tree or for fully balanced trees. We also note that for larger trees the throughput distribution for a random tree is skewed towards the fully balanced case, which is associated with GPU saturation at these problem sizes. Finally, we note that pectinate-case performance is approximately twice as fast with the P5000 GPU under CUDA as compared to the R9 Nano GPU using our OpenCL implementation. Effective performance towards the pectinate end of the tree symmetry scale remains highly relevant as phylogenetic inference programs are optimized such that only a

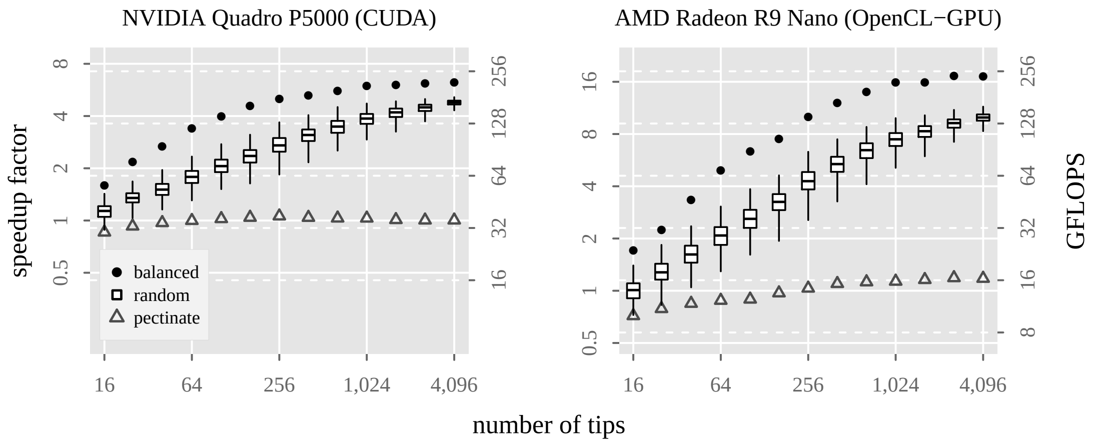


Figure 5.4: Plots showing throughput for the partial likelihood kernel with subtree concurrency for fully balanced trees (black dots), for 1,000 random topology trees (distribution characterized by box plot), and for pectinate trees (open triangles) for a problem with 512 site patterns and increasing number of tips for two GPU device/framework pairs. Left-axis *speedup factor* indicates performance gain relative to the average pectinate tree throughput. Speedup factors, throughput, and number of tips are on a log-scale.

subtree representing the modified portion of the overall tree is recomputed for each topology change. These subtrees are often much less balanced than the full tree.

5.3.3 Application-Level Results

We used our adapted version of MrBayes 3.2.6 to assess application-level performance gains for our concurrency work across a variety of parallel computing devices. For these benchmarks we used a dataset with 500 taxa and 759 unique site patterns of *rbc*L, the chloroplast gene encoding the large subunit of ribulose-1,5-bisphosphate carboxylase/oxygenase, which is derived from a study of angiosperm relationships [117]. We partitioned the sequence data based on codon position, resulting in 3 subsets with 253 unique site patterns each, and inferences were run using the MrBayes default single-precision floating point format.

We chose a dataset with a high number of sequences and with few patterns, further broken into independent subsets, to best showcase the gains in concurrency described in this paper. Previously problems with these characteristics have been the most challenging for effective parallelization. BEAGLE-enabled MrBayes peak performance for datasets with many more patterns and using higher state-count models are reported elsewhere [12, 35].

Speedups for this challenging MrBayes analysis improve as we enable partition and subtree concurrency, across all hardware resources and corresponding frameworks (Fig. 5.5 on the next page). We observe an average speedup gain of 1.5-fold for subtree concurrency and 1.4-fold for partition concurrency across all hardware devices. For the best performing resource (NVIDIA Quadro P5000 GPU with CUDA) we observe a 1.7-fold gain in speedup when using both concurrency improvements, ultimately resulting in a 10-fold speedup over the native MrBayes SSE run time.

We have attempted but were unable to compare our work to the most recent proposals from other authors for parallel MrBayes acceleration. For aMC^3 [13], which proposes an adaptive multi-GPU approach, we were unable to perform any analyses with the publicly available code due to execution errors. Additionally, aMC^3 is based on MrBayes 3.1.2 which lacks several features and converges more slowly than version 3.2 [4], making it unsuitable for a direct comparison to our work. For sMC^3 [77], which proposes more efficient CPU + GPU parallelism and reports

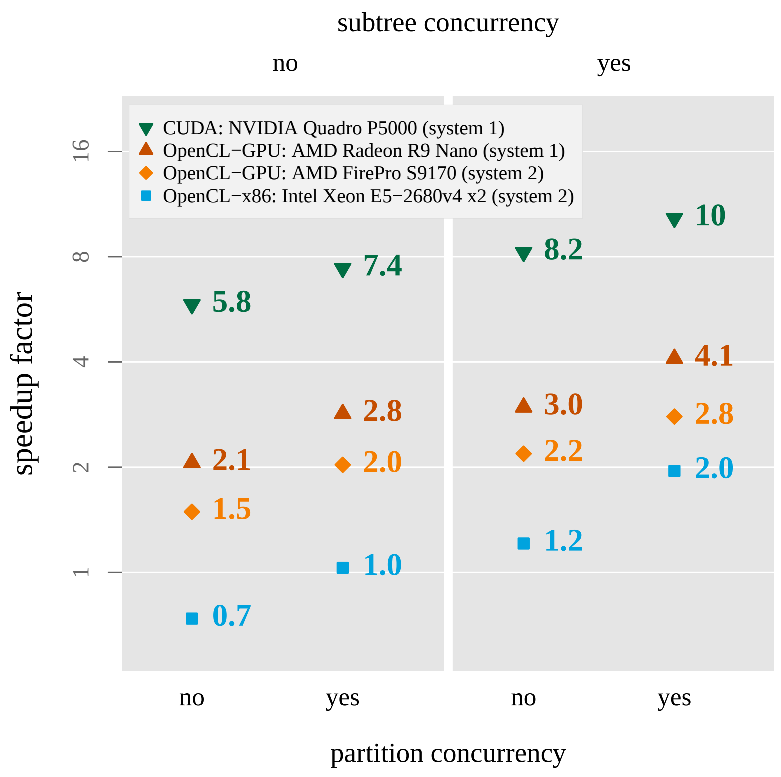


Figure 5.5: Performance gains for a MrBayes nucleotide-model analysis for various hardware platforms when using the BEAGLE library, with and without partition and subtree concurrency. Speedup factors are relative to the total run time when using the standard MrBayes SSE likelihood calculator and are shown on a log-scale.

speedups over previous versions of BEAGLE, neither the source code nor a binary file appear to be readily available.

5.4 Conclusion

Enabling further concurrency of computation in BEAGLE as described here allows a wider range of phylogenetic inferences to benefit from parallel computing hardware. Analyses with many small data subsets or with large trees but few site patterns, now benefit from increased throughput on multi and manycore resources. This work represents an important step in combining the capabilities of increasingly parallel hardware, and the demands of progressively more sophisticated phylogenetic inference analyses.

Acknowledgments

We thank Marc Suchard, University of California, Los Angeles, and Andrew Rambaut, University of Edinburgh; Mark Berger, NVIDIA; and Greg Stoner and Ben Sander, AMD. This work was supported by the National Science Foundation grant numbers DBI-0755048 and DBI-1356562.

Bibliography

[1] Guy Baele and Philippe Lemey. Bayesian evolutionary model testing in the phylogenomics era: matching model complexity with computational efficiency. *Bioinformatics*, 29(16):1970–1979, 2013.

[2] Derrick J. Zwickl. *Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion.* PhD thesis, University of Texas, Austin (TX), 2006.

[3] Paul O. Lewis. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution*, 15(3):277–83, 1998.

[4] Fredrik Ronquist, Maxim Teslenko, Paul van der Mark, Daniel L. Ayres, Aaron Darling, Sebastian Höhna, Bret Larget, Liang Liu, Marc A. Suchard, and John P. Huelsenbeck. MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space. *Systematic biology*, 61(3):539–542, 2012.

[5] Alexei J. Drummond, Marc A. Suchard, Dong Xie, and Andrew Rambaut. Bayesian phylogenetics with BEAUti and the BEAST 1.7. *Molecular Biology and Evolution*, 29:1969–1973, 2012.

[6] Joseph Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–76, 1981.

[7] Joseph Felsenstein. The number of evolutionary trees. *Systematic Biology*, 27(1):27–33, 1978.

[8] NVIDIA Corporation. CUDA C Programming Guide 8.0. `http://docs.nvidia.com/cuda/cuda-c-programming-guide`, 2017.

[9] Frederico Pratas, Pedro Trancoso, Alexandros Stamatakis, and Leonel Sousa. Fine-grain parallelism using multi-core, Cell/BE, and GPU systems: Accelerating the phylogenetic likelihood function. In *ICPP'09. International Conference on Parallel Processing, 2009.*, pages 9–17. IEEE, 2009.

[10] Frederico Pratas, Pedro Trancoso, Leonel Sousa, Alexandros Stamatakis, Guochun Shi, and Volodymyr Kindratenko. Fine-grain parallelism using multicore, Cell/BE, and GPU systems. *Parallel Computing*, 38(8):365–390, 2012.

[11] Marc A. Suchard and Andrew Rambaut. Many-core algorithms for statistical phylogenetics. *Bioinformatics*, 25(11):1370–1376, 2009.

[12] Daniel L. Ayres, Aaron Darling, Derrick J. Zwickl, Peter Beerli, Mark T. Holder, Paul O. Lewis, John P. Huelsenbeck, Fredrik Ronquist, David L. Swofford, Michael P. Cummings, Andrew Rambaut, and Marc A. Suchard. BEAGLE: An application programming interface and high-performance computing library for statistical phylogenetics. *Systematic Biology*, 61(1):170–173, 2012.

[13] Jie Bao, Hongju Xia, Jianfu Zhou, Xiaoguang Liu, and Gang Wang. Efficient implementation of MrBayes on multi-GPU. *Molecular Biology and Evolution*, 30(6):1471, 2013.

[14] Cheng Ling, Tsuyoshi Hamada, Jianing Bai, Xianbin Li, Douglas Chesters, Weimin Zheng, and Weifeng Shi. MrBayes tgMC 3: A tight GPU implementation of MrBayes. *PloS ONE*, 8(4):e60667, 2013.

[15] Shuai Pang, Rebecca J Stones, Ming-Ming Ren, Xiao-Guang Liu, Gang Wang, Hong-ju Xia, Hao-Yang Wu, Yang Liu, and Qiang Xie. GPU MrBayes V3.1: MrBayes on graphics processing units for protein sequence data. *Molecular Biology and Evolution*, 32(9):2496–2497, 2015.

[16] Nikolaos Alachiotis, Euripides Sotiriades, Apostolos Dollas, and Alexandros Stamatakis. Exploring FPGAs for accelerating the phylogenetic likelihood function. In *IPDPS 2009. IEEE International Symposium on Parallel & Distributed Processing, 2009.*, pages 1–8. IEEE, 2009.

[17] Nikolaos Alachiotis, Alexandros Stamatakis, Euripides Sotiriades, and Apostolos Dollas. A reconfigurable architecture for the phylogenetic likelihood function. In *FPL 2009. International Conference on Field Programmable Logic and Applications, 2009.*, pages 674–678. IEEE, 2009.

[18] Frederico Pratas and Leonel Sousa. *Applying the Stream-Based Computing Model to Design Hardware Accelerators: A Case Study*, pages 237–246. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[19] Stephanie Zierke and Jason D. Bakos. FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods. *BMC Bioinformatics*, 11(1):184, 2010.

[20] Xizhou Feng, Duncan A. Buell, John R. Rose, and Peter J. Waddell. Parallel algorithms for Bayesian phylogenetic inference. *Journal of Parallel and Distributed Computing*, 63(7–8):707 – 718, 2003. Special Issue on High-Performance Computational Biology.

[21] Xizhou Feng, Kirk W. Cameron, and Duncan A. Buell. PBPI: a high performance implementation of Bayesian phylogenetic inference. In *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, page 40. IEEE, 2006.

[22] Jianfu Zhou, Gang Wang, and Xiaoguang Liu. *A New Hybrid Parallel Algorithm for MrBayes*, pages 102–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[23] Andre J Aberer, Kassian Kobert, and Alexandros Stamatakis. ExaBayes: massively parallel Bayesian tree inference for the whole-genome era. *Molecular Biology and Evolution*, 31(10):2553–2556, 2014.

[24] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[25] Wilfred K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[26] Charles J. Geyer. Markov chain Monte Carlo maximum likelihood. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, pages 156–163, Fairfax, Virginia, 1991. Interface Foundation of North America.

[27] Gautam Altekar, Sandhya Dwarkadas, John P. Huelsenbeck, and Fredrik Ronquist. Parallel Metropolis-coupled Markov chain Monte Carlo for Bayesian phylogenetic inference. Technical Report TR784, Department of Computer Science, University of Rochester, July 2002.

[28] Gautam Altekar, Sandhya Dwarkadas, John P. Huelsenbeck, and Fredrik Ronquist. Parallel Metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference. *Bioinformatics*, 20(3):407–415, 2004.

[29] Jianfu Zhou, Xiaoguang Liu, Douglas S. Stones, Qiang Xie, and Gang Wang. MrBayes on a graphics processing unit. *Bioinformatics*, 27(9):1255–1261, 2011.

[30] J. Felsenstein. Confidence intervals on phylogenies: an approach using the bootstrap. *Evolution*, 39:783–791, 1985.

[31] David L. Swofford. *PAUP*: Phylogenetic Analysis using Parsimony (* and Other Methods), Version 4.* Sinauer Associates, Sunderland (MA), 2003.

[32] Alexandros Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–90, Nov 2006.

[33] Adam L. Bazinet, Derrick J. Zwickl, and Michael P. Cummings. A gateway for phylogenetic analysis powered by grid computing featuring GARLI 2.0. *Systematic Biology*, 63(5):812–818, 2014.

[34] Andrew Gelman and Donald B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical science*, pages 457–472, 1992.

[35] Daniel L. Ayres and Michael P. Cummings. Heterogeneous hardware support in BEAGLE, a high-performance computing library for statistical phylogenetics. In *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*, Bristol, UK, in press.

[36] Daniel L. Ayres and Michael P. Cummings. Configuring concurrent computation of phylogenetic partial likelihoods: Accelerating analyses using the BEAGLE library. In *2017 17th International Conference on Algorithms and Architectures for Parallel Processing: ICA3PP Collocated Workshops*, Helsinki, Finland, in press.

[37] Stéphane Guindon, Jean-François Dufayard, Vincent Lefort, Maria Anisimova, Wim Hordijk, and Olivier Gascuel. New algorithms and methods to estimate maximum-likelihood phylogenies: Assessing the performance of PhyML 3.0. *Systematic Biology*, 59(3):307–321, 2010.

[38] Nuno R. Faria, Andrew Rambaut, Marc A. Suchard, Guy Baele, Trevor Bedford, Melissa J. Ward, Andrew J. Tatem, João D. Sousa, Nimalan Arinaminpathy, Jacques Pépin, David Posada, Martine Peeters, Oliver G. Pybus, and Philippe Lemey. The early spread and epidemic ignition of HIV-1 in human populations. *Science*, 346(6205):56–61, 2014.

[39] Tiago Gräf, Bram Vrancken, Dennis Maletich Junqueira, Rúbia Marília de Medeiros, Marc A. Suchard, Philippe Lemey, Sabrina Esteves de Matos Almeida, and Aguinaldo Roberto Pinto. Contribution of epidemiological predictors in unraveling the phylogeographic history of HIV-1 subtype C in Brazil. *Journal of Virology*, 89(24):12341–12348, 2015.

[40] Martin Hoenigl, Antoine Chaillon, Harald H. Kessler, Bernhard Haas, Evelyn Stelzl, Karin Weninger, Susan J. Little, and Sanjay R. Mehta. Characterization of HIV transmission in South-East Austria. *PLoS One*, 11(3):e0151478, 2016.

[41] Avram Levy, Jason Roberts, Jurissa Lang, Simone Tempone, Alison Kesson, Alfred Dofai, Andrew J. Daley, Bruce Thorley, and David J. Speers. Enterovirus D68 disease and molecular epidemiology in Australia. *Journal of Clinical Virology*, 69:117–121, 2015.

[42] Tavis K. Anderson, Brian A. Campbell, Martha I. Nelson, Nicola S. Lewis, Alicia Janas-Martindale, Mary Lea Killian, and Amy L. Vincent. Characterization of co-circulating swine influenza A viruses in North America and the

identification of a novel H1 genetic clade with antigenic significance. *Virus Research*, 201:24–31, 2015.

[43] Jessica Hedge, Samantha J. Lycett, and Andrew Rambaut. Real-time characterization of the molecular epidemiology of an influenza pandemic. *Biology Letters*, 9(5):20130331, 2013.

[44] Sarah C. Hill, Youn-Jeong Lee, Byung-Min Song, Hyun-Mi Kang, Eun-Kyoung Lee, Amanda Hanna, Marius Gilbert, Ian H. Brown, and Oliver G. Pybus. Wild waterfowl migration and domestic duck density shape the epidemiology of highly pathogenic H5N8 influenza in the Republic of Korea. *Infection, Genetics and Evolution*, 34:267–277, 2015.

[45] Daniel Magee, Rachel Beard, Marc A. Suchard, Philippe Lemey, and Matthew Scotch. Combining phylogeography and spatial epidemiology to uncover predictors of H5N1 influenza A virus diffusion. *Archives of Virology*, 160(1):215–224, 2015.

[46] Cara C. Burns, Jing Shaw, Jaume Jorba, David Bukbuk, Festus Adu, Nicksy Gumede, Muhammed Ali Pate, Emmanuel Ade Abanida, Alex Gasasira, Jane Iber, Qi Chena, Annelet Vincenta, Paul Chenoweth, Elizabeth Hendersona, Kathleen Wannemuehlerb, Asif Naeemh, Rifqiyah Nur Umamih, Yorihiro Nishimurah, Hiroyuki Shimizuh, Marycelin Babac, Adekunle Adenijid, A. J. Williamsa, David R. Kilpatricka, M. Steven Oberstea, Steven G. Wassilakb, Oyewale Tomorii, Mark A. Pallanscha, and Olen Kew. Multiple independent emergences of type 2 vaccine-derived polioviruses during a large outbreak in northern Nigeria. *Journal of Virology*, 87(9):4907–4922, 2013.

[47] Maia A. Rabaa, Cameron P. Simmons, Annette Fox, Mai Quynh Le, Thuy Thi Thu Nguyen, Hai Yen Le, Robert V. Gibbons, Xuyen Thanh Nguyen, Edward C. Holmes, and John G. Aaskov. Dengue virus in sub-tropical northern and central Viet Nam: population immunity and climate shape patterns of viral invasion and maintenance. *PLoS Neglected Tropical Diseases*, 7(12):e2581, 2013.

[48] M. R. Nunes, Gustavo Palacios, Nuno Rodrigues Faria, Edivaldo Costa Sousa Jr., Jamilla A. Pantoja, Sueli G. Rodrigues, Valeria L. Carvalho, D. B. Medeiros, Nazir Savji, Guy Baele, Marc A. Suchard, Philippe Lemey, Pedro F. C. Vasconcelos, and W. Ian Lipkin. Air travel is associated with intracontinental spread of Dengue virus serotypes 1–3 in Brazil. *PLoS Neglected Tropical Diseases*, 8(4):e2769, 2014.

[49] C Torres, C Lema, F Gury Dohmen, F Beltran, L Novaro, S Russo, MC Freire, A Velasco-Villa, VA Mbayed, and DM Cisterna. Phylodynamics of vampire bat-transmitted rabies in Argentina. *Molecular Ecology*, 23(9):2340–2352, 2014.

[50] Yung-Cheng Lin, Pei-Yu Chu, Mei-Yin Chang, Kuang-Liang Hsiao, Jih-Hui Lin, and Hsin-Fu Liu. Spatial temporal dynamics and molecular evolution of re-emerging rabies virus in Taiwan. *International journal of molecular sciences*, 17(3):392, 2016.

[51] Mark Zeller, Celeste Donato, Nídia Sequeira Trovão, Daniel Cowley, Elisabeth Heylen, Nicole C. Donker, John K. McAllen, Asmik Akopov, Ewen F. Kirkness, Philippe Lemey, Marc van Ranst, Jelle Matthijnssens, and Carl D. Kirkwood. Genome-wide evolutionary analyses of G1P [8] strains isolated before and after rotavirus vaccine introduction. *Genome Biology and Evolution*, 7(9):2473–2483, 2015.

[52] Andriyan Grinev, Caren Chancey, Evgeniya Volkova, Germán Añez, Daniel A. R. Heisey, Valerie Winkelman, Gregory A. Foster, Phillip Williamson, Susan L. Stramer, and Maria Rios. Genetic variability of West Nile Virus in US blood donors from the 2012 epidemic season. *PLoS Neglected Tropical Diseases*, 10(5):e0004717, 2016.

[53] Simon Rasmussen, Morten Erik Allentoft, Kasper Nielsen, Ludovic Orlando, Martin Sikora, Karl-Göran Sjögren, Anders Gorm Pedersen, Mikkel Schubert, Alex van Dam, Christian Moliin Outzen Kapel, Henrik Bjørn Nielsen, Søren Brunak, Pavel Avetisyan, Andrey Epimakhov, Mikhail Viktorovich Khalyapin, Artak Gnuni, Aviar Kriiska, Irena Lasak, Mait Metspalu, Vyacheslav Moiseyev, Andrei Gromov, Dalia Pokutta, Lehti Saag, Liivi Varul, Levon Yepiskoposyan, Thomas Sicheritz-Pontén, Robert A. Foley, Marta Mirazón Lahr, Rasmus Nielsen, Kristian Kristiansen, and Eske Willerslev. Early divergent strains of *Yersinia pestis* in Eurasia 5,000 years ago. *Cell*, 163(3):571–582, 2015.

[54] BH Wiseman, ED Fountain, MH Bowie, S He, and RH Cruickshank. Vivid molecular divergence over volcanic remnants: the phylogeography of *Megadromus guerinii* on Banks Peninsula, New Zealand. *New Zealand Journal of Zoology*, pages 1–12, 2016.

[55] Toru Katoh, Hiroyuki F. Izumitani, Shinji Yamashita, and Masayoshi Watada. Multiple origins of Hawaiian drosophilids: Phylogeography of *Scaptomyza* Hardy (Diptera: Drosophilidae). *Entomological Science*, 2016.

[56] Stephen W. Attwood, Guan-Nan Huo, and Jian-Wen Qiu. Update on the distribution and phylogenetics of *Biomphalaria* (Gastropoda: Planorbidae) populations in Guangdong Province, China. *Acta Tropica*, 141:258–270, 2015.

[57] Michael J. Andersen, Carl H. Oliveros, Christopher E. Filardi, and Robert G. Moyle. Phylogeography of the variable dwarf-kingfisher *Ceyx lepidus* (Aves: Alcedinidae) inferred from mitochondrial and nuclear DNA sequences. *The Auk*, 130(1):118–131, 2013.

[58] Trevor D. Price, Daniel M. Hooper, Caitlyn D. Buchanan, Ulf S. Johansson, D. Thomas Tietze, Per Alstrom, Urban Olsson, Mousumi Ghosh-Harihar,

Farah Ishtiaq, Sandeep K. Gupta, Jochen Martens, Bettina Harr, Pratap Singh, and Dhananjai Mohan. Niche filling slows the diversification of Himalayan songbirds. *Nature*, 509(7499):222–225, 2014.

[59] Michael J. Andersen, Peter A. Hosner, Christopher E. Filardi, and Robert G. Moyle. Phylogeny of the monarch flycatchers reveals extensive paraphyly and novel relationships within a major Australo-Pacific radiation. *Molecular Phylogenetics and Evolution*, 83:118–136, 2015.

[60] Rupert A. Collins, Ralf Britz, and Lukas Rüber. Phylogenetic systematics of leaffishes (Teleostei: Polycentridae, Nandidae). *Journal of Zoological Systematics and Evolutionary Research*, 53(4):259–272, 2015.

[61] Edward D. Burress. Ecological diversification associated with the pharyngeal jaw diversity of neotropical cichlid fishes. *Journal of Animal Ecology*, 85(1):302–313, 2016.

[62] Arley Camargo, Fernanda P. Werneck, Mariana Morando, Jack W. Sites, and Luciano J. Avila. Quaternary range and demographic expansion of *Liolaemus darwinii* (Squamata: Liolaemidae) in the Monte Desert of Central Argentina using Bayesian phylogeography and ecological niche modelling. *Molecular Ecology*, 22(15):4038–4054, 2013.

[63] Julia Naumann, Karsten Salomo, Joshua P. Der, Eric K. Wafula, Jay F. Bolin, Erika Maass, Lena Frenzke, Marie-Stéphanie Samain, Christoph Neinhuis, and Stefan Wanke. Single-copy nuclear genes place haustorial Hydnoraceae within Piperales and reveal a Cretaceous origin of multiple parasitic angiosperm lineages. *PLoS ONE*, 8:e79204, 2013.

[64] Ivana Rešetnik, Zlatko Satovic, Gerald M. Schneeweiss, and Zlatko Liber. Phylogenetic relationships in Brassicaceae tribe Alysseae inferred from nuclear ribosomal and chloroplast DNA sequence data. *Molecular Phylogenetics and Evolution*, 69(3):772–786, 2013.

[65] Alex D. Twyford and Jannice Friedman. Adaptive divergence in the monkey flower *Mimulus guttatus* is maintained by a chromosomal inversion. *Evolution*, 69(6):1476–1486, 2015.

[66] Anna V. Williams, Laura M. Boykin, Katharine A. Howell, Paul G. Nevill, and Ian Small. The complete sequence of the *Acacia ligulata* chloroplast genome reveals a highly divergent clpP1 gene. *PloS One*, 10(5):e0125768, 2015.

[67] Dafu Ru, Kangshan Mao, Lei Zhang, Xiaojuan Wang, Zhiqiang Lu, and Yongshuai Sun. Genomic evidence for polyphyletic origins and interlineage gene flow within complex taxa: a case study of *Picea brachytyla* in the Qinghai-Tibet Plateau. *Molecular ecology*, 25(11):2373–86, Jun 2016.

[68] Kim Schwarze, Abhilasha Singh, and Thorsten Burmester. The full globin repertoire of turtles provides insights into vertebrate globin evolution and functions. *Genome Biology and Evolution*, 7(7):1896–1913, 2015.

[69] Kim Rohlfing, Friederike Stuhlmann, Margaret F. Docker, and Thorsten Burmester. Convergent evolution of hemoglobin switching in jawed and jawless vertebrates. *BMC evolutionary biology*, 16:30, 2016.

[70] M. A. Miller, W. Pfeiffer, and T. Schwartz. Creating the CIPRES science gateway for inference of large phylogenetic trees. In *Gateway Computing Environments Workshop (GCE), 2010*, pages 1–8, nov. 2010.

[71] Jun Chai, Huayou Su, Mei Wen, Xing Cai, Nan Wu, and Chunyuan Zhang. Resource-efficient utilization of CPU/GPU-based heterogeneous supercomputers for Bayesian phylogenetic inference. *The Journal of Supercomputing*, 66(1):364–380, 2013.

[72] Z. Jin and J. Bakos. Extending the BEAGLE library to a multi-FPGA platform. *BMC Bioinformatics*, 14(1):25, 2013.

[73] Filip Bielejec, Philippe Lemey, Luiz Max Carvalho, Guy Baele, Andrew Rambaut, and Marc A. Suchard. π BUSS: a parallel BEAST/BEAGLE utility for sequence simulation under complex evolutionary scenarios. *BMC Bioinformatics*, 15(1):1, 2014.

[74] T. Flouri, F. Izquierdo-Carrasco, D. Darriba, A. J. Aberer, L-T Nguyen, B. Q. Minh, A. von Haeseler, and A. Stamatakis. The phylogenetic likelihood library. *Systematic Biology*, 64(2):356–362, 2015.

[75] Cheng Ling, Chunbao Zhou, Arong Luo, Guoguang Zhao, Tsuyoshi Hamada, and Xiaoyan Zhu. Optimizing the Bayesian inference of phylogeny on graphic processors. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 333–342. IEEE, 2015.

[76] Chunbao Zhou, Xianyu Lang, Yangang Wang, and Chaodong Zhu. gPGA: GPU accelerated population genetics analyses. *PloS one*, 10(8):e0135028, 2015.

[77] Lídia Kuan, Frederico Pratas, Leonel Sousa, and Pedro Tomás. MrBayes sMC3: Accelerating Bayesian inference of phylogenetic trees. *The International Journal of High Performance Computing Applications*, 2016.

[78] Fredrik Ronquist and John P. Huelsenbeck. MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–4, 2003.

[79] Alexei J. Drummond and Andrew Rambaut. BEAST: Bayesian evolutionary analysis by sampling trees. *BMC Evolutionary. Biology*, 7:214, 2007.

[80] Jerome Regier, Jeffrey Shultz, Andreas Zwick, April Hussey, Bernard Ball, Regina Wetzer, Joel Martin, and Clifford Cunningham. Arthropod relationships revealed by phylogenomic analysis of nuclear protein-coding sequences. *Nature*, 463:1079–1083, Feb 2010.

[81] Bob Mau and Michael A. Newton. Phylogenetic inference for binary data on dendograms using Markov chain Monte Carlo. *Journal of Computational and Graphical Statistics*, 6(1):122–131, 1997.

[82] Ziheng Yang and Bruce Rannala. Bayesian phylogenetic inference using DNA sequences: a Markov chain Monte Carlo method. *Molecular biology and evolution*, 14(7):717–724, 1997.

[83] Bret Larget and Donald L. Simon. Markov chain Monte Carlo algorithms for the Bayesian analysis of phylogenetic trees. *Molecular biology and evolution*, 16(6):750–759, 1999.

[84] Bob Mau, Michael A. Newton, and Bret Larget. Bayesian phylogenetic inference via Markov chain Monte Carlo methods. *Biometrics*, 55(1):1–12, 1999.

[85] John P. Huelsenbeck and Fredrik Ronquist. MrBayes: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17(8):754–755, 2001.

[86] Bruce Rannala and Ziheng Yang. Phylogenetic inference using whole genomes. *Annual Review of Genomics and Human Genetics*, 9:217–231, 2008.

[87] Fredrik Ronquist and Andrew R. Deans. Bayesian phylogenetics and its influence on insect systematics. *Annual review of entomology*, 55, 2010.

[88] Clemens Lakner, Paul van der Mark, John P. Huelsenbeck, Bret Larget, and Fredrik Ronquist. Efficiency of Markov chain Monte Carlo tree proposals in Bayesian phylogenetics. *Systematic biology*, 57(1):86–103, 2008.

[89] Sebastian Höhna and Alexei J. Drummond. Guided tree topology proposals for Bayesian phylogenetic inference. *Systematic biology*, 61(1):1–11, 2012.

[90] Gareth O. Roberts and Jeffrey S. Rosenthal. Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009.

[91] John P. Huelsenbeck, Bret Larget, and David L. Swofford. A compound poisson process for relaxing the molecular clock. *Genetics*, 154(4):1879–1892, 2000.

[92] Jeffrey L. Thorne and Hirohisa Kishino. Divergence time and evolutionary rate estimation with multilocus data. *Systematic biology*, 51(5):689–702, 2002.

[93] Thomas Lepage, David Bryant, Hervé Philippe, and Nicolas Lartillot. A general comparison of relaxed molecular clock models. *Molecular biology and evolution*, 24(12):2669–2680, 2007.

[94] Alexei J. Drummond, Simon Y. W. Ho, Matthew J. Phillips, and Andrew Rambaut. Relaxed phylogenetics and dating with confidence. *PLoS biology*, 4(5):e88, 2006.

[95] Tanja Gernhard. The conditioned reconstructed process. *Journal of theoretical biology*, 253(4):769–778, 2008.

[96] Tanja Stadler. On incomplete sampling under birth-death models and connections to the sampling-based coalescent. *Journal of theoretical biology*, 261(1):58–66, 2009.

[97] Sebastian Höhna, Tanja Stadler, Fredrik Ronquist, and Tom Britton. Inferring speciation and extinction rates under different sampling schemes. *Molecular biology and evolution*, 28(9):2577–2589, 2011.

[98] Scott V. Edwards, Liang Liu, and Dennis K. Pearl. High-resolution species trees without concatenation. *Proceedings of the National Academy of Sciences*, 104(14):5936–5941, 2007.

[99] Liang Liu and Dennis K. Pearl. Species trees from gene trees: reconstructing Bayesian posterior distributions of a species phylogeny using estimated gene tree distributions. *Systematic biology*, 56(3):504–514, 2007.

[100] Nick Goldman. Statistical tests of models of DNA substitution. *Journal of molecular evolution*, 36(2):182–198, 1993.

[101] David Posada and Keith A. Crandall. Modeltest: testing the model of DNA substitution. *Bioinformatics*, 14(9):817–818, 1998.

[102] David Posada. jModelTest: phylogenetic model averaging. *Molecular biology and evolution*, 25(7):1253–1256, 2008.

[103] Marc A. Suchard, Robert E. Weiss, and Janet S. Sinsheimer. Bayesian selection of continuous-time Markov chain evolutionary models. *Molecular biology and evolution*, 18(6):1001–1013, 2001.

[104] John P. Huelsenbeck, Bret Larget, and Michael E. Alfaro. Bayesian phylogenetic model selection using reversible jump Markov chain Monte Carlo. *Molecular biology and evolution*, 21(6):1123–1133, 2004.

[105] Michael A. Newton and Adrian E. Raftery. Approximate Bayesian inference with the weighted likelihood bootstrap. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 3–48, 1994.

[106] Nicolas Lartillot and Hervé Philippe. Computing Bayes factors using thermodynamic integration. *Systematic biology*, 55(2):195–207, 2006.

[107] Wangang Xie, Paul O. Lewis, Yu Fan, Lynn Kuo, and Ming-Hui Chen. Improving marginal likelihood estimation for Bayesian phylogenetic model selection. *Systematic biology*, 60(2):150–160, 2011.

[108] Fredrik Ronquist, Seraina Klopfstein, Lars Vilhelmsen, Susanne Schulmeister, Debra L. Murray, and Alexandr P. Rasnitsyn. A total-evidence approach to dating with fossils, applied to the early radiation of the Hymenoptera. *Systematic Biology*, 61(6):973–999, 2012.

[109] Gregory Frederick Diamos, Andrew Robert Kerr, Sudhakar Yalamanchili, and Nathan Clark. Ocelot: A dynamic optimization framework for bulk-synchronous applications in heterogeneous systems. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pages 353–364, New York, NY, USA, 2010. ACM.

[110] Rodrigo Domínguez, Dana Schaa, and David Kaeli. Caracal: Dynamic translation of runtime environments for gpus. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-4, pages 5:1–5:7, New York, NY, USA, 2011. ACM.

[111] John A. Stratton, Sam S. Stone, and Wen-mei W. Hwu. MCUDA: An efficient implementation of CUDA kernels for multi-core CPUs. In José Nelson Amaral, editor, *Languages and Compilers for Parallel Computing: 21th International Workshop, LCPC 2008, Edmonton, Canada, July 31 - August 2, 2008, Revised Selected Papers*, pages 16–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[112] Gabriel Martinez, Mark Gardner, and Wu-chun Feng. CU2CL: A CUDA-to-OpenCL translator for multi-and many-core architectures. In *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, pages 300–307, 2011.

[113] Mark Gardner, Paul Sathre, Wu-chun Feng, and Gabriel Martinez. Characterizing the challenges and evaluating the efficacy of a CUDA-to-OpenCL translator. *Parallel Computing*, 39(12):769–786, 2013.

[114] Matthew J. Harvey and Gianni De Fabritiis. Swan: A tool for porting CUDA programs to OpenCL. *Computer Physics Communications*, 182(4):1093 – 1099, 2011.

[115] C++ Standards Committee and others. ISO International Standard ISO/IEC 14882: 2011, Programming Language C++. Technical report, Geneva, Switzerland, 2011.

[116] Adam L. Bazinet, Michael P. Cummings, Kim T. Mitter, and Charles W. Mitter. Can RNA-Seq resolve the rapid radiation of advanced moths and butterflies (Hexapoda: Lepidoptera: Apoditrysia)? An exploratory study. *PLoS ONE*, 8(12):e82615, Dec 2013.

[117] Mark W. Chase, Douglas E. Soltis, Richard G. Olmstead, David Morgan, Donald H. Les, Brent D. Mishler, Melvin R. Duvall, Robert A. Price, Harold G. Hills, Yin-Long Qiu, Gregory M. Plunkett, Pamela S. Soltis, Susan M.

Swensen, Stephen E. Williams, Paul A. Gadek, Christopher J. Quinn, Luis E. Eguiarte, Edward Golenberg, Gerald H. Learn, Jr., Sean W. Graham, Spencer C. H. Barrett, S. Dayanandan, and Victor A. Albert. Phylogenetics of seed plants: an analysis of nucleotide sequences from the plastid gene *rbc*L. *Annals of the Missouri Botanical Garden*, pages 528–580, 1993.