2018 IEEE International Parallel and Distributed Processing Symposium Workshops

Rerooting Trees Increases Opportunities for Concurrent Computation and Results in Markedly Improved Performance for Phylogenetic Inference

Daniel L. Ayres and Michael P. Cummings
Center for Bioinformatics and Computational Biology
University of Maryland
College Park, MD, 20742, USA
ayres@umiacs.umd.edu and mike@umiacs.umd.edu

***Abstract*—Parallel computing algorithms benefit from increases in concurrency when the hardware capacity is being under utilized. For likelihood-based molecular evolution inferences this can be due to small problem sizes, or because hardware capacity has increased beyond dataset sizes. A central concept in this domain is a bifurcating tree, which represents evolutionary relationships. The topology of the tree being evaluated directly affects the degree of parallelism that can be exploited by likelihood-based algorithms. For time-reversible evolutionary models we can reroot an unbalanced tree in order to make it more symmetric, without affecting the likelihood. Based on the reduction in number of concurrent operation sets, we define a best-case theoretical expectations, based on tree size and topology, for speedup due to rerooting which approaches 2-fold as the number of tip nodes increases for pectinate trees, and much higher values for some random topologies as the number of tip nodes increases. Empirical results using the NVIDIA CUDA implementation of the BEAGLE library confirm the merit of this approach. For pectinate trees we observe speedups of up to 1.93-fold due to rerooting and even larger speedups for random trees for the core likelihood-evaluation function in BEAGLE.**

***Keywords*-Bayes methods; Biology computing; Evolution (biology); Phylogeny; Maximum likelihood estimation; Multicore processing; Parallel programming**

I. Introduction

The process of adapting existing computational methods to parallel architectures benefits from a broad perspective on concurrency, the simultaneous execution of independent operations. Opportunities exist for increased parallel performance where the number of concurrent operations is less than the capacity of the available device(s) (e.g., achieved occupancy is less than theoretical occupancy for a CUDA device). More forward-thinking perspectives include developing and employing methods to increase concurrency even when capacity of present devices is exceeded (e.g., occupancy is at theoretical limits or at levels practically achievable) such that concurrency, and hence performance, will increase via strong scaling for future devices with increased capacity. Such perspectives are particularly beneficial in the context of mixed problems sizes, where performance improvements from increased concurrency may immediately be realized for smaller problems sizes, which is the case for computing of phylogenetic likelihoods in evolutionary biology (Section II). Furthermore, advancing computational technology characterized by increasing device capacity and improving memory performance may shift the demarcation between compute- and memory-bound properties for specific problems, and thus provide further potential for realizing performance gains via concurrent computation.

Initial work on the BEAGLE library for high-performance statistical phylogenetic inference [1] focused on fine-grained parallelism where each character (i.e., 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 characteristics 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 the level of sequence position via the likelihood calculation (Section II-A), parallelism at this level is a logical focus in pursuit of improved performance.

More recently we have broadened our efforts to include medium-grained parallelism, by seeking higher-level independence where concurrent computation opportunities afford further application of the fine-grained parallelism capabilities of the BEAGLE library. We identified independent likelihood estimates in analyses of partitioned datasets and in proposed trees [2], and configured concurrent computation of these likelihoods via CUDA and OpenCL frameworks [3]. In the work presented here we extend this medium-grained parallelism to additional opportunities for concurrent computation of independent partial likelihoods arrays in the statistical phylogenetic setting, and characterize the potential for increased performance.

The paper continues by providing some domain science context (Section II), brief description of the BEAGLE library (Section III), review of independent likelihood estimates in analyses of partitioned datasets and in proposed trees (Section IV), and describe in detail additional opportunities for concurrent computation of independent partial likelihoods

978-1-5386-5555-9/18/$31.00 ©2018 IEEE
DOI 10.1109/IPDPSW.2018.00049

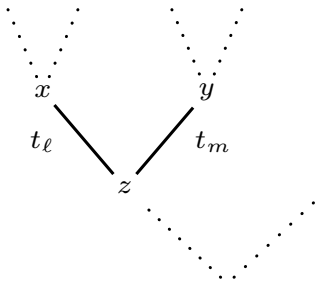

Figure 1. Likelihood subtree to which the core likelihood calculation applies. Solid lines depict focus subtree branches, dotted lines contextual branches.

arrays for rerooted trees (Section V).

II. Evolutionary Biology, the Domain Science

A major component of evolutionary biology research involves inference of relationships of species, genes, or alleles. In the phylogenetics or gene genealogical setting these entities are often referred to more generically as operational taxonomic units (OTUs). Research in evolutionary biology can generally be divided in either of two broad categories:

1) *macroevolution*, which involves the processes of speciation and extinction where OTUs are typically species or genes representing species; and
2) *microevolution*, which involves the processes affecting changes in the genetic structure of populations where OTUs are alleles of genes or other genetic data.

Phylogenetics, in the broad sense, 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.

Population genetics includes research objectives of estimation of size, growth rate, migration, and other parameters characterizing populations. Modern population genetics is based on coalescent theory [4]–[6], which represents a retrospective approach in that its conceptual framework is a binary tree, a gene genealogy, representing the relationships of alleles going back in time. Gene lineages sequentially unite — coalesce — ultimately to a common ancestor. Well-developed mathematical theory provides expectations for the timing of these coalescent events, which can be used to estimate population genetic parameters.

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. Consequently they also share a computational bottleneck, the calculation of the likelihood values.

A. Likelihood Function

Statistical phylogenetics comprises maximum likelihood estimation and Bayesian analysis, and is established as the most effect methods for inferring both phylogenetic trees and gene genealogies. Heuristic algorithms are employed to search through the space of possibilities to find a putatively optimal solution (maximum likelihood) or characterize posterior probability distributions (Bayesian analysis). Computation in statistical phylogenetics is dominated by calculation of the likelihood of trees [7]. For example, profiling GARLI [8], a leading phylogenetic inference program, demonstrates that for DNA models, computing likelihood calculations requires in excess of 94% of the overall run time. More complex models, such as those based on amino-acid or codon models, are often even more computationally intensive. Hence the focus on decreasing the time required for calculation of the likelihood function as a means to increase the performance of statistical inference-based phylogenetic and population genetic analyses.

A subtree comprising a parent node, z, two child nodes, x and y, and connecting branches of length, t_ℓ and t_m (Fig. 1) is the focus of the core partial likelihood calculation. The calculation is iterated for all such such subtrees within the larger tree required for the analysis. The partial likelihood function is as follows [7]:

$$L_k^{(i)}(z) = \left(\sum_x \Pr(x|z,t_\ell) L_\ell^{(i)}(x) \right) \times \left(\sum_y \Pr(y|z,t_m) L_m^{(i)}(y) \right) \quad (1)$$

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 OTUs (e.g., species, alleles). Additionally the tree search space is very large; the number of unrooted topologies possible for n OTUs is given by the double factorial function $(2n-5)!!$ [9]. 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.

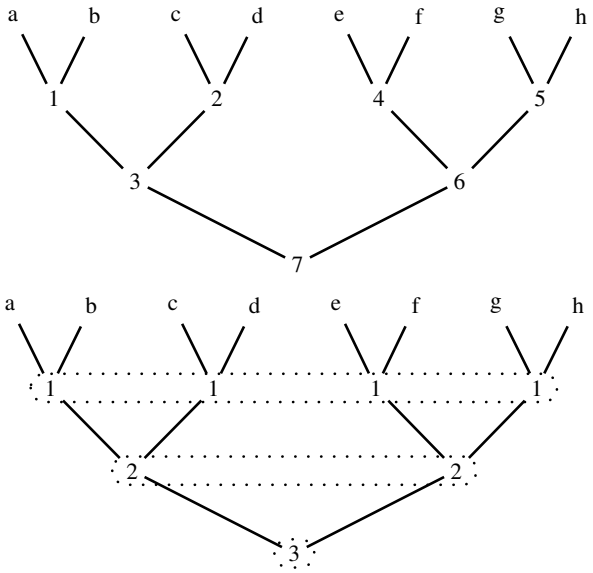

Figure 2. Upper: Example tree of eight OTUs (labeled a–h) with post-order traversal and subtree calculations in series corresponding to node numbers, with $n - 1 = 7$ subtree calculations. Lower: Same tree with reverse level-order, or breadth-first, traversal. Concurrent subtree calculations possible for independent nodes designated with corresponding node numbers and enclosed by dotted lines, with $\lceil log_2 n \rceil = 3$ sets.

III. The BEAGLE Library and Application Programming Interface

The BEAGLE library and application programming interface (API) [1] is a parallel computing platform for high-performance calculation of phylogenetic likelihoods. BEAGLE comprises a collection of efficient implementations using a shared code design employing CUDA and OpenCL frameworks [3], and hardware-specific optimizations to exploit a wide-range of hardware parallelism including CPU and Xeon Phi, vectorization intrinsics (e.g., SSE, AVX), and GPUs. BEAGLE also defines a uniform API that facilitates its integration with host (calling) programs. It is the first and most widely adopted library for phylogenetic likelihood calculation, having been integrated into popular phylogenetics software including BEAST [10], MrBayes [11], and PhyML [12]. Consequently BEAGLE has been used extensively for phylogenetic analyses.

IV. Independent Partial Likelihood Operations

Much of our previous effort on parallelization of the phylogenetic likelihood function (Eq. II-A) has focused on fine-grained concurrency by developing efficient algorithms with atomization at the levels of positions and states, essentially computing $p \times s$ (sequence patterns $\times$ character states) as a 2-dimensional grid. Recently we have further increased concurrency by exploiting the fact that many analyses involve data subsets, so called partitioned analyses, for which phylogenetic partial likelihoods can be calculated independently. Similarly, we exploit the fact that many subtrees (Fig. 1) are autonomous in that their associated partial likelihoods can be calculated independently within the larger tree of which they are constituent parts. We describe each of these medium-grained concurrency exploits briefly in the following two subsections.

A. Pattern Partition Concurrency

A popular approach to phylogenetic analyses is to partition sequence data into subsets, often based on genes or codon positions, and allowing independent model parameters for each of these different subsets. The resulting model flexibility improves overall model fit, and has proven to be an effective means of obtaining improved results. The independence of these data subsets provides an opportunity for increased parallelization, as likelihood calculations for each subset can be computed concurrently.

We have implemented pattern partition concurrency in two approaches. The first is as a set of streams in CUDA or queues in OpenCL, and the second through a multi-operation kernel. For the later implementation we modified our earlier partial likelihood kernel in CUDA to compute multiple likelihood arrays in a single execution launch, and use pointer arithmetic to allow different input and output arrays for different execution blocks. Further details regarding pattern partition concurrency and the resulting performance improvements are provided elsewhere [2].

B. Independent Subtree Concurrency

The number of subtrees requiring calculation for any full tree is $n - 1$, again, where n is the number of OTUs (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. 2, upper). But, again, many of these subtrees are autonomous and likelihoods for each can be calculated concurrently. However, to realize any potential concurrency related to autonomous subtrees present in a given tree, operations need to be sent to BEAGLE following a reverse level-order, or breadth-first, traversal of the tree being evaluated. 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. 2, lower). This subtree concurrency can be conceptualized as a 3-dimensional grid, $p \times s \times subtrees$, and implemented as a single kernel launch, or alternatively as a set of streams in CUDA or queues in OpenCL [2]. The performance improvements resulting from computing subtrees concurrently is substantial [2].

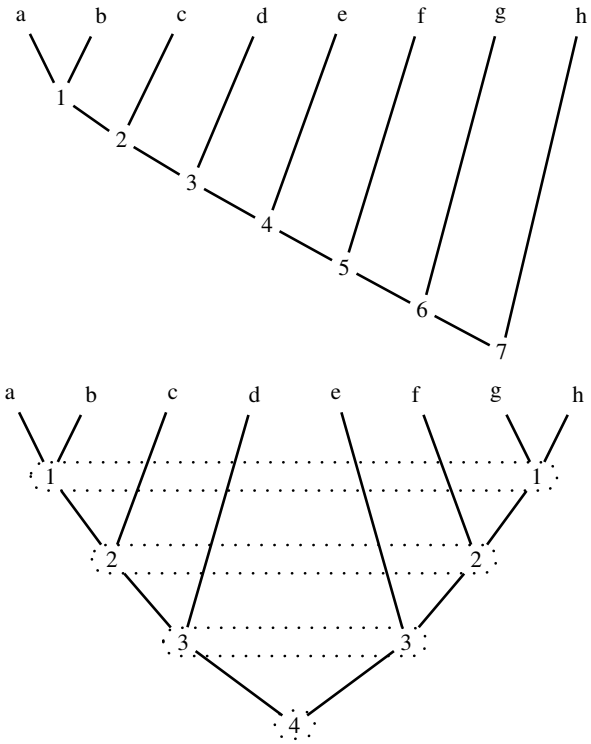

Figure 3. Upper: Example pectinate tree of eight OTUs (labeled a–h) with traversal and subtree calculations in series corresponding to node numbers, with $n - 1 = 7$ subtree calculations. Lower: Rerooted version of same tree to maximize opportunities for concurrent subtree calculations designated and with corresponding node numbers enclosed by dotted lines, with $\lceil n/2 \rceil = 4$ sets.

V. Rerooted Trees and Independent Partial Likelihood Operations

A. Pectinate Trees

In the case of a fully pectinate tree no subtrees are autonomous, or so it might initially appear (Fig. 3, top). Note that for time-reversible evolutionary models, which are the most common employed, the likelihood of the tree is independent of the location of the root [7]. We can use this property to our advantage by rerooting the tree to enable additional concurrent operations.

For pectinate trees evaluated with a time-reversible evolutionary model, the number of autonomous subtrees can be maximized if the tree is rerooted so that $\lfloor n/2 \rfloor$, or alternatively, $\lceil n/2 \rceil$, tips are on one side of the root. Such rerooting and tree evaluation with a reverse level-order traversal results in $\lceil n/2 \rceil$ sets of concurrent operations (Fig. 3, bottom). Consequently for pectinate trees we have a precise expectation for relative performance increase resulting from optimal rerooting and concurrent computation of independent subtrees, apart from the cost of rerooting itself, possible inefficiencies, and stochastic variance. The expectation is that as $n \uparrow, (n-1)/\lceil n/2 \rceil \to 2$, i.e., the performance gain for should be $2 - \epsilon$ fold, $\epsilon > 0$.

B. Intermediate Trees (Neither Fully Balanced Nor Fully Pectinate)

For any tree, rooted or unrooted, evaluated with a time-reversible evolutionary model, the number of autonomous subtrees can be maximized if the tree is rerooted so that $\lfloor n/2 \rfloor$, or alternatively, $\lceil n/2 \rceil$, tips are on one side of the root. Thus, more generally, any optimally rerooted tree results in $\leq \lceil n/2 \rceil$ sets of concurrent operations. Therefore, with appropriate algorithmic capability and sufficient hardware capacity the number of sets of subtree calculations can be very substantially reduced from $n - 1$ for standard post-order traversal to a value in the interval $[\lceil log_2 n \rceil, \lceil n/2 \rceil]$ with post-order traversal and concurrent computation.

As in the case of pectinate trees, we can specify the expectation for relative performance resulting from optimal rerooting and concurrent computation of independent subtrees, apart from the cost of rerooting itself, possible inefficiencies, and stochastic variance. This expectation is that the performance gain should fall in the interval $[(n-1)/\lceil n/2 \rceil, (n-1)/\lceil log_2 n \rceil]$, i.e., the performance gain should be between $2 - \epsilon$ fold, $\epsilon > 0$, and $(n-1)/\lceil log_2 n \rceil$ fold, depending on the balance-pectinate properties of the tree and the initial rooting.

VI. Methods

We have developed a set of benchmarks to assess performance gains from increased concurrency due to rerooting. These benchmarks were run on a top-of-the-line workstation and use an extended version of the testing platform which is part of the BEAGLE library source code.

A. Concurrent Partial Likelihood Computation in BEAGLE

The current development version of BEAGLE can concurrently compute independent partial likelihood arrays on parallel hardware devices. We leveraged this capability to achieve performance increases through optimal rerooting. BEAGLE employs a variety of methods to concurrently compute partial likelihoods, depending on a combination of parameters including problem size, hardware, and framework [2] (herein described as an *implementation class*).

For this work, we focused on the NVIDIA CUDA implementation for problems with fewer than 10^3 alignment patterns. Problems with few patterns have more scope to benefit from increased concurrency and, generally, we have noticed that the CUDA implementation to be the most efficient in terms of framework overhead.

For this implementation class, BEAGLE concurrently computes partial likelihoods using a *multi-operation kernel*, which enables the computation of multiple likelihood arrays in a single execution launch. This is done using pointer arithmetic to allow different input and output arrays for different execution blocks. For this to work BEAGLE requires

Table I
SYSTEM SPECIFICATIONS.

	System 1
CPU	Intel Xeon E5-2697v4 (x2)
GPU	NVIDIA Quadro GP100
Linux kernel	3.10.0
GCC version	6.2.0
CUDA release	8.0.61

partial likelihood subtree operations to be sent according to a reverse level-order traversal of the proposed tree. BEAGLE adds each consecutive operation to a set until it finds an operation that is dependent on the result of a previous operation in the set. The library then starts a new operation set, repeating the same process. Once all operations are processed in this manner, operation sets are successively launched for concurrent computation using a *multi-operation* partial likelihoods kernel [2].

B. System Specifications

We report benchmark results for the system configuration shown in Table I. We focused on the NVIDIA CUDA platform as it previously has shown least amount of overhead [3], thus reducing the amount of noise for the empirical evaluation of the gains from rerooting. Further, we utilized a top-of-the-line Pascal generation GPU which uses a GP100 chip with 3,584 CUDA cores and HBM2 memory with 720 GB/s of total bandwidth. This allowed us to push the hardware saturation point for parallelism further into larger problem sizes.

C. Performance Metric

We have used the performance of the partial likelihoods kernel in BEAGLE as the relevant metric throughout this study, as rerooting increases concurrency of computation for this function only. Using this metric allows us to best focus on the specific performance effect of rerooting. Further, the partial likelihoods kernel is the computational bottleneck for phylogenetic analyses and performance improvements to this function correspond directly to application run time gains for full inferences [2], [3].

Specifically, 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 II-A). In contrast to a direct timing benchmark, throughput allows us to more easily compare performance across different problem sizes and to assess how efficiently the hardware resource is being utilized.

D. Tree Topologies

To better assess performance gains due to rerooting across a range of scenarios, we augmented *synthetictest* (a testing program included in the BEAGLE repository, see Section VI-F) to be able to generate additional tree topology types. By default *synthetictest* generates trees that are fully balanced, that is, trees that have the optimal topology type for exploiting partial likelihood concurrency and thus that do not benefit from rerooting.

We developed new topology-type options to enable the generation of pectinate, and arbitrary or random topology trees, in addition to the default balanced topology. These additional options allow us to assess the effect of rerooting on worst and average-case topologies for concurrent computation.

For randomly generating a topology, we iteratively construct trees one tip node at a time. We connect each new tip to a randomly chosen sibling, which can be any of the existing nodes, including internal ones. The new tip node and the randomly chosen sibling node then gain a new parent node, which becomes a child of the previous parent node of the sibling. For pectinate trees we use the same procedure but always use the current root as the sibling node as each tip node is added.

E. Rerooting

We extended *synthetictest* to support rerooting of any tree such that it is optimally balanced, and thus requires the fewest number of parallel kernel launches for computing its likelihood. We implemented rerooting as a one-time procedure, performed before any calls to the BEAGLE library, so that the effectiveness of this operation did not impact the benchmarks in this study.

To perform an optimally balanced rerooting, we use a naive algorithm that exhaustively searches all possible rootings. For each branch of the original tree, we recursively reconstruct a tree with a new root at this branch. Then, for each of these tree rootings, we assess the number of necessary kernel launches to compute its likelihood. This is done using a reverse level-order traversal, and counting the number of sets of independent partial likelihood operations. We then choose a rooting that results in the fewest number of concurrent operation sets.

F. Test Program and Scripts

We used the BEAGLE test program *synthetictest* and a set of Python scripts to evaluate the performance effect of rerooting on the partial likelihoods function. The *synthetictest* program can generate arbitrary datasets, evolutionary models, and tree topologies according to user-defined parameters and uses the BEAGLE library to evaluate the overall tree likelihood. This test program is included with the library source code, available at https://github.com/beagle-dev/beagle-lib.

The results shown in the next section can be reproduced using *synthetictest* with a combination of the command line settings described in Table II. The command line options shown are divided in two categories, those which were used across all benchmarks and those which were used according to the specific tree type being evaluated.

Table II
TEST PROGRAM PARAMETERS.

Command line option	Description
Always used	
`--rsrc n`	selects the hardware resource
`--taxa n`	sets the number of taxa or OTUs in the randomly generated data set
`--sites n`	sets the number of site patterns in the randomly generated data set
`--reps n`	sets the number of calculation repetitions
`--full-timing`	enables output of detailed timing information
`--manualscale`	enables application-managed floating-point rescaling
`--rescale-frequency n`	sets the frequency of computation of new rescaling factors, relative to the number of calculation repetitions
Benchmark dependent	
`--pectinate`	sets tree topology type to pectinate
`--randomtree`	sets tree topology type to arbitrary
`--reroot`	enables optimal rerooting of tree
`--seed n`	sets the random seed to be used for generating arbitrary alignment data, evolutionary model parameters, and tree topology

For this study, the `--rsrc` option was used to select the GP100 GPU on *System 1* (Table I) for all benchmarks. The `--taxa` option was set according to the tree size being benchmarked. The `--sites` option was set to 512 for all problems; we used a relatively small alignment size to avoid saturating the GPU when computing the partial likelihood at a single node, thus allowing gains from concurrent computation of multiple nodes (see previous work [3] for a GPU performance curve relative to number of sites). The `--reps` option was set to $1,000$ to overcome a potential warm-up period from the GPU and capture best-case performance. The `--full-timing` option was enabled to capture both compute throughput from the partial likelihood kernel and the number of concurrent operation launches necessary to compute the tree likelihood. The `--manualscale` option was enabled to overcome underflow due to the use of single-precision floating-point format for trees with large numbers of taxa in this study (it was enabled for all benchmarks, to allow direct comparison across problem sizes). The `--rescale-frequency` option was set to $1,000$ so that new rescaling factors were only computed once per run and thus did not affect measurement of best-case performance.

Newly developed `--pectinate` and `--randomtree` options were used to enable different tree topology types, according to the benchmark. Balanced topology trees were generated by omitting either of these options. The `--reroot` option was used in combination with either of the two non-balanced topology type options, to optimally reroot the tree. The `--seed` option was used in conjunction `--randomtree` option and set to different values (1 to $1,000$) to generate trees with different, arbitrary, topologies.

As an example, to evaluate the performance of a randomly generated, optimally rooted tree with 64 OTUs, we would have used the following command:

```
./synthetictest --rsrc 1 --taxa 64
--sites 512 --reps 1000 --full-timing
--manualscale --rescale-frequency 1000
--randomtree --reroot --seed 1.
```

Python scripts were used to automate the process of initiating multiple runs of the *synthetictest* program, each with different command line parameters. The scripts were also used to collect the program output from these runs and process it into data tables.

VII. RESULTS

Here we describe our investigations of the performance effect of the concurrency gains due to rerooting topologies. We have used the *synthetictest* program to benchmark the performance of the core likelihood function in the BEAGLE library v3.0 (upcoming release) when evaluating pectinate trees, perfectly balanced trees, random trees, rerooted pectinate trees and rerooted random trees. System specifications were as shown in Table I.

A. Rerooting Effect on Number of Concurrent Operation Sets

Here we explore how optimal rerooting reduces the number of required concurrent operation sets (i.e., the number of kernel launches for the GPU implementation in BEAGLE) for computing the overall tree likelihood.

Figure 4 shows how, for a sample of 100 arbitrarily generated trees each with 256 OTUs, the number of necessary operation sets is consistently reduced by performing an optimally balanced rerooting. For this sample of trees, we observe that the number of operation sets is reduced by as much as half for trees that were originally less balanced (i.e., required more operation sets). We also observe that in one case, for a tree which required 26 operation sets, rerooting did not offer any benefit as this tree was already optimally balanced. Overall, it is clear that the rerooting procedure can significantly reduce the required number of concurrent operation sets, which can lead to performance gains if the parallel hardware resource has unutilized capacity.

B. Rerooting Effect on Throughput Performance

We used the same data set of 100 trees from Section VII-A to measure the effect of rerooting on throughput performance of the partial likelihoods GPU kernel in BEAGLE.

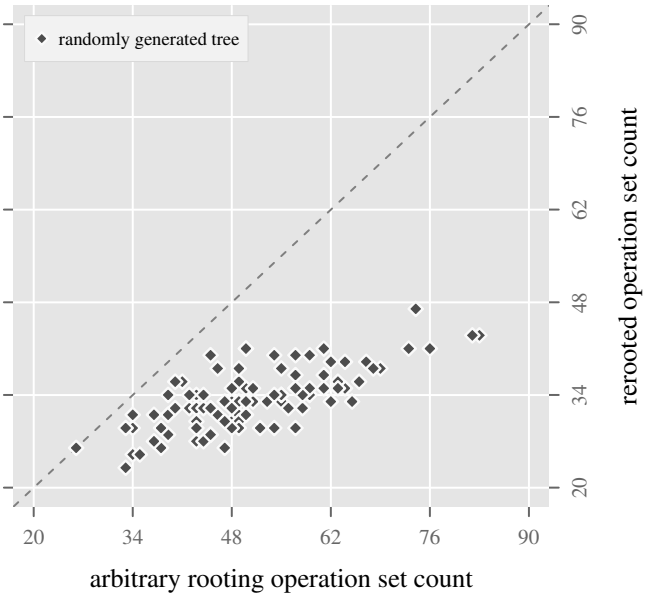

Figure 4. Plot showing the number of required GPU kernel launches of the partial likelihood function with subtree concurrency for 100 randomly generated trees with arbitrary rooting, and for the same trees with optimal rerooting. Each tree had 256 OTUs. Dashed line represents the null hypothesis of no difference in kernel launches resulting from rerooting.

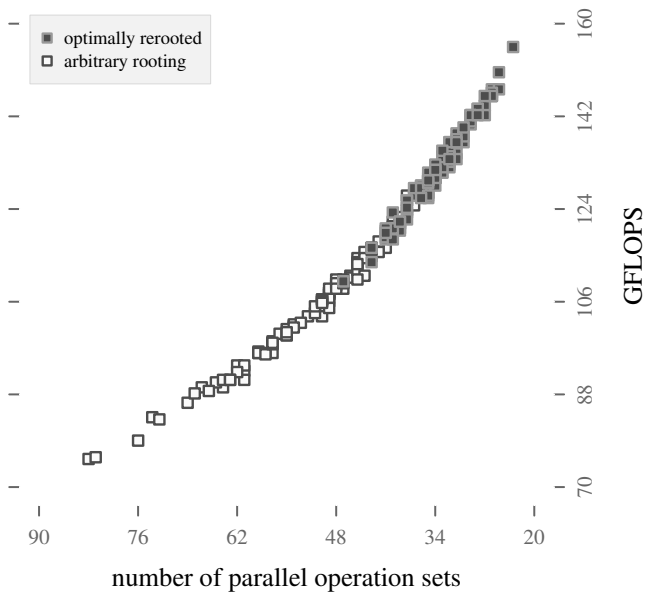

Figure 5. Plot showing throughput for the partial likelihood kernel with subtree concurrency for 100 randomly generated trees with arbitrary rooting, and for the same trees with optimal rerooting for a problem with 512 site patterns and 256 OTUs. Benchmarks were performed using the GP100 GPU on *System 1*. Note that the horizontal axis is decreasing from left to right, this was done to facilitate the visualization of the increase in performance with the decrease in number of operations.

Table III
PROPORTION OF THEORETICAL SPEEDUP REALIZED FOR CONCURRENT SUBTREE CALCULATIONS FOR 64 OTUS AND 512 PATTERNS.

Topology Type	Theoretical Expectation	NVIDIA GP100	Realized Speedup
balanced	10.5	3.95	0.38
pectinate	1.00	1.00	na
pectinate rerooted	1.97	1.55	0.79
random	[1.85, 5.25]	[1.56, 3.07]	[0.84, 0.58]
random rerooted	[3.15, 5.73]	[2.22, 3.30]	[0.70, 0.57]

Each sample in the data set had 512 unique site patterns. This experiment allows us to analyze in more detail the performance effect of rerooting on a fixed-size tree with varying topology.

Figure 5 shows how throughput performance increases with the inverse of the number of concurrent operation sets, as we would expect due to better GPU utilization. Further, since rerooting increases the number of concurrent operations, the result is higher throughput. For this sample of trees, we observe a mean performance improvement of 1.26-fold due to rerooting.

C. Theoretical Comparison

Here we compare the previously established theoretical bounds to empirical results for speedups of the partial-likelihoods kernel relative to the sequential case when using concurrent subtree operations. To perform the sequential benchmarks, we modified the BEAGLE source code to disable multi-operation kernel launches, so that each partial likelihood array was computed in turn.

Theoretical speedup expectations for balanced and pectinate tree topologies were previously defined in Section V. For randomly generated tree topologies we determined the theoretical speedup bounds for each specific tree in the random sample being evaluated by counting the number of operation sets. The theoretical bounds together with empirical results allow us to further assess rerooting gains, as we compare across a variety of tree topology types. These comparisons also allow us to assess the effectiveness of the GPU implementation in BEAGLE, and of the hardware and software solution as a whole.

Table III compares empirical speedups to theoretical expectations across a variety of tree topology types, for a tree with 64 OTUs and 512 unique site patterns. For the random topology benchmarks we used samples of 100 arbitrarily generated trees. We observe that, as expected, none of the empirical results fall outside of the theoretical bounds. The proportion of theoretical maximum speedup realized ranges 0.38 (for a balanced topology) to 0.84 (for random

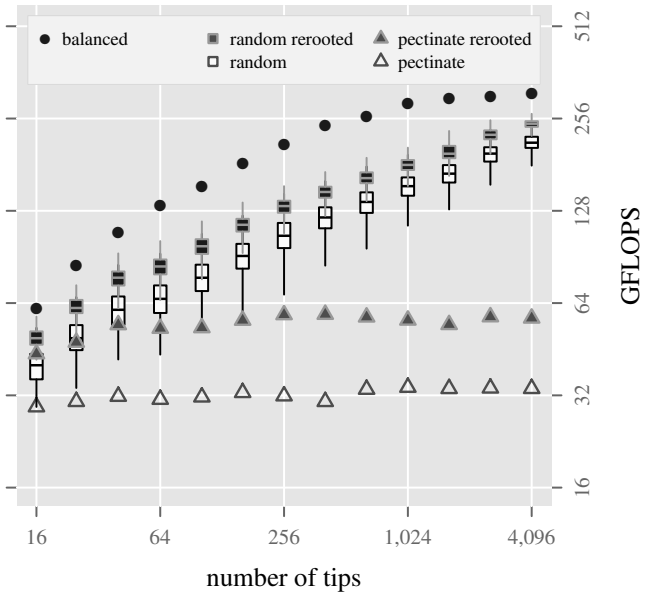

Figure 6. Plot showing throughput for the partial likelihood kernel with subtree concurrency for fully balanced trees (filled dots), for 1,000 random topology trees (distribution characterized by open box plot), and for pectinate trees (open triangles) for a problem with 512 site patterns and increasing number of OTUs. Throughput for rerooted trees are shown in filled box plots (random topology trees) and filled triangles (pectinate trees). Throughput and number of tips are on a log-scale. Benchmarks were performed using the GP100 GPU on *System 1*.

topology), and thus much of the expected performance gains due to rerooting are realized in the empirical results.

D. Overall Effect of Rerooting on Throughput Performance

Here we expand on the benchmarks performed for Section VII-B by including balanced and pectinate tree topologies, using larger samples of 1,000 random trees each, and using a range of tree sizes, from 16 to 4,096 OTUs. Figure 5 shows throughput performance with concurrent computation of independent subtrees for a problem with 512 patterns across a variety of tree sizes and topology types, with and without rerooting. For reference, we note that the non-rerooted pectinate case (open white triangle) is equivalent to the performance for any tree topology when computing partial likelihood arrays without subtree concurrency (i.e., the prevailing methodology).

We observe that rerooting pectinate trees consistently results in significant increases in performance, with a best-case speedup of 1.93-fold for a tree with 406 OTUs. Effective performance towards the pectinate end of the tree symmetry scale is highly relevant as phylogenetic inference programs are optimized such that only a subtree representing the modified portion of the overall tree is recomputed for each topology change. These subtrees are often less symmetrical than the full tree.

Further, we can note that randomly generated trees also consistently benefit from rerooting, although to a lesser extent than pectinate ones. Overall, we observe increasing speedups with tree size for non-pectinate trees. We also note that for larger trees the throughput distribution for a random tree is skewed towards the fully balanced case, which is attributable to larger random trees having relatively more inherent concurrent computing opportunities on one hand, and on the other hand hardware device saturation decreasing performance for fully balanced trees relatively more strongly with increased tree size.

VIII. Discussion

For simplicity of study design, we have explored here rerooting performance gains only in synthetic tests and have not empirically assessed the applicability of our approach in the course of a complete inference run. There are important factors that need to be considered when analyzing the relevance of the results presented here to the performance of a full inference.

Firstly, we have considered likelihood calculations only in the context of a full tree traversal. However, many phylogenetic analysis programs only calculate those subtree partial likelihoods as required for a topology change, which in some cases is a small part of the full tree, as other subtrees may not require recalculation. This elimination of unnecessary computation is among the reasons for the increased performance of modern statistical phylogenetics programs. Hence it maybe fair to ask, does the concurrent computation of autonomous subtrees (Section IV-B) results in performance gains in practice with a modern phylogenetics analysis program?

Additionally, our design separated the rerooting steps from the concurrent computation of tree likelihoods, and we did not consider the computational cost of the rerooting operation itself. Our use of a naive exhaustive search through all possible rootings to find the optimal rooting (Section VI-E) was done for expedience, and a more efficient algorithm could be employed. Regardless of the specific algorithm, the desired outcome is that the benefit gained by the increased concurrent computation on the rerooted tree is greater than the cost of optimally rerooting the original tree. The specific characteristic of this cost-benefit relationship is also an important issue when considering the applicability of our approach.

In this section we consider in detail the impact of these factors on how well our results might translate to real-world gains. We conclude that we can expect the benefits of the rerooting approach presented here to largely apply in the context of a phylogenetic analysis.

A. Applicability Considerations

In this study we have measured the effect of balanced rerooting on the performance of the partial likelihood func-

tion in BEAGLE. Although focusing on this metric allows us to directly investigate the throughput impact of rerooting, ultimately our objective is to allow phylogenetic analysis to complete in less time. We considered an application-side implementation of rerooting to be beyond the scope of this initial study and thus do not present empirical results of application-level performance. Nonetheless the results presented here together with our previous research indicate that, at least for large trees with few site patterns, balanced rerooting would result in clear performance gains at the application level.

In order to reach this conclusion, we considered the typical characteristic of the following factors related to the performance of phylogenetic inferences: 1) the proportion of time spent computing partial likelihoods; 2) the topology of the tree being updated in a given iteration; 3) the relative cost and necessary frequency of the rerooting operation.

In regards to factor 1, the time spent computing partial likelihoods, our experience has been that this function is where the vast majority of the run time is spent. We have performed profiling using BEAST [10], MrBayes [11], and GARLI [8] and have observed that, for nucleotide-model analyses, this proportion is typically in excess of 0.9 of the overall run time. Further, in previous studies where we have benchmarked throughput for the partial likelihoods function in BEAGLE, we have observed that improvements to the performance of this function directly correspond to application-level gains [2], [3].

Although it is clear that computing partial likelihoods is the primary bottleneck for typical phylogenetic inference analyses, it is also important to consider factor 2, the topology of the tree or subtree being updated at each iteration of the search algorithm of the inference program. For iterations which involve a topology change, modern inference programs only recompute the partial likelihoods for a small subset of the overall tree. In contrast, for an otherwise underutilized GPU, the potential rerooting gains described in this study increase as tree size increases, as larger trees allow for more nodes to be computed in parallel.

This fact might appear to limit the applicability of the approach described here, however there are additional factors to consider. For a given inference algorithm, search iterations that change a non-topology parameter will often require recomputation of the entire tree. For these iterations, the results shown here will directly correspond to the expected gains. Further, recent developments in Bayesian phylogenetic inference use an adaptive Markov Chain Monte Carlo (MCMC) approach to allow for multiple continuous parameters to be updated in a single iteration, in order to increase the effective sample size [13]. One consequence relevant here is that, for programs using this adaptive MCMC approach, every non-topology move requires updating the entire tree. Although this technique is currently only used for rooted trees with BEAST, it is not incompatible with unrooted trees.

Still further in regard to factor 2, our own empirical testing with MrBayes has shown that the general approach of exploiting partial likelihood concurrency of independent nodes on a proposed tree results in appreciable application-level speedups [2], even if the calling program only recomputes a small subtree on topology changes. Specifically, for a unpartitioned data set with 500 taxa and 759 unique patterns, we observed an overall speedup of $1.41\times$ on an NVIDIA Quadro P5000 GPU by enabling concurrent computation of partial likelihoods on independent nodes. These benchmarks were performed with a random starting tree and we expect that applying the balanced rerooting described here to the starting topology would have resulted in further performance gains.

Finally, we consider factor 3, the relative cost and required frequency of the rerooting operation. For this study, we did not explore the computational cost or performance impact of the rerooting operation itself. Most importantly we believe that it is sufficient to reroot the starting tree in order to achieve appreciable performance gains. This is because, in aggregate, we can expect topology moves to be randomly distributed on either side of a balanced rooting. Given that typical phylogenetic inferences take many hours or days to complete, the cost of finding a balanced rooting and applying it to the starting tree will be trivial in comparison. Nonetheless, we also consider it likely that further balanced rerootings, later in the search process, might result in further performance gains, and this remains an issue to be studied.

IX. Conclusion

Parallel computing algorithms for calculating tree likelihoods, such as those employed by the BEAGLE library, benefit from balanced topologies. These topologies allow for greater concurrency and thus better utilization of the multi-core hardware. Non-balanced trees can be rerooted to make them more symmetric, thus reducing the number of concurrent operation sets required for further likelihood evaluations.

Empirical results demonstrate that rerooting can lead to significant increases in performance for the core likelihood function in BEAGLE, with speedups on a modern GPU approaching 2-fold for pectinate trees with more than 10^2 OTUs and even larger performance gains for some random tree topologies.

We expect that, for similarly sized problems, the performance gains observed here can be largely realized by phylogenetic inference applications which use BEAGLE once they incorporate rerooting using a run-time efficient algorithm.

Acknowledgment

We thank three anonymous reviewers, and Mark Berger, NVIDIA. This work was supported by the US National Science Foundation grant numbers DBI-1356562 and DBI-1661443.

REFERENCES

[1] D. L. Ayres, A. Darling, D. J. Zwickl, P. Beerli, M. T. Holder, P. O. Lewis, J. P. Huelsenbeck, F. Ronquist, D. L. Swofford, M. P. Cummings, A. Rambaut, and M. A. Suchard, "BEAGLE: An application programming interface and high-performance computing library for statistical phylogenetics," *Syst. Biol.*, vol. 61, pp. 170–173, 2012.

[2] D. L. Ayres and M. P. Cummings, "Configuring concurrent computation of phylogenetic partial likelihoods: Accelerating analyses using the BEAGLE library," in *Algorithms and Architectures for Parallel Processing. ICA3PP 2017, Helsinki, Finland*, ser. Lect. Notes Comput. Sc., S. Ibrahim, K. Choo, Z. Yan, and W. Pedrycz, Eds., vol. 10393, August 2017, pp. 533–547.

[3] ——, "Heterogeneous hardware support in BEAGLE, a high-performance computing library for statistical phylogenetics," in *46th International Conference on Parallel Processing Workshops (ICPPW 2017)*, Bristol, United Kingdom, August 2017, pp. 23–32.

[4] J. F. C. Kingman, "The coalescent," *Stoch. Proc. Appl.*, vol. 13, no. 3, pp. 235–248, 1982.

[5] ——, "Exchangeability and the evolution of large populations," in *Exchangeability in Probability and Statistics*, ser. Proceedings of the International Conference on Exchangeability in Probability and Statistics, G. Koch and F. Spizzichino, Eds. Amsterdam: North-Holland Elsevier, April 1982, pp. 97–112.

[6] ——, "On the genealogy of large populations," *J. Appl. Probab.*, pp. 27–43, 1982.

[7] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach." *J. Mol. Evol.*, vol. 17, no. 6, pp. 368–76, 1981.

[8] D. J. Zwickl, "Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion," Ph.D. dissertation, University of Texas, Austin (TX), 2006.

[9] J. Felsenstein, "The number of evolutionary trees," *Syst. Zoo.*, vol. 27, no. 1, pp. 27–33, 1978.

[10] A. J. Drummond, M. A. Suchard, D. Xie, and A. Rambaut, "Bayesian phylogenetics with BEAUti and the BEAST 1.7," *Mol. Biol. Evol.*, vol. 29, pp. 1969–1973, 2012.

[11] F. Ronquist, M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Höhna, B. Larget, L. Liu, M. A. Suchard, and J. P. Huelsenbeck, "MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space," *Syst. Biol.*, vol. 61, no. 3, pp. 539–542, 2012.

[12] S. Guindon, J.-F. Dufayard, V. Lefort, M. Anisimova, W. Hordijk, and O. Gascuel, "New algorithms and methods to estimate maximum-likelihood phylogenies: Assessing the performance of PhyML 3.0," *Syst. Biol.*, vol. 59, no. 3, pp. 307–321, 2010.

[13] G. Baele, P. Lemey, A. Rambaut, and M. A. Suchard, "Adaptive MCMC in Bayesian phylogenetics: an application to analyzing partitioned data in BEAST," *Bioinformatics*, 2017.