University Of Victoria
Faculty of Engineering
Spring 2012 - CSC 464 Project Report

Concurrent Trees

An Investigation on Parallelizing DNAml, an aging Maximum Likelihood Tree Generation program

Jeremy Ho
Stephen Tredger

April 26 2012

CSC 464 - Yvonne Coady

Problem

Phylogenetic trees are a visual representation of how species or genes are evolutionarily related to each other. Typically the length of a branch in the tree represents the amount of time that has passed before the species diverged from one of its ancestors. The topology of the tree gives insight into how all the species are related to each other. This type of information is useful in the field of Bioinformatics because it provides the basis for further research into the relation of the targeted species.

Generating phylogenetic trees can be done with a variety of algorithms including UPGMA, Neighbor Joining, Maximum Parsimony, Maximum Likelihood, and Markov Chain Monte Carlo based Bayesian Inference. Each of these algorithms have their strengths and weaknesses when generating phylogenetic trees as compared to each other. The comparison of these algorithms is outside of this paper's scope, as we focus specifically on the Maximum Likelihood algorithm.

Maximum Likelihood (ML) is a statistical model that infers a phylogenetic tree based on the likelihood of a proposed model to be valid given an input set. ML is generally considered to create very accurate trees and is a relatively robust against violations to the evolutionary model [1]. However, although ML phylogeny reconstruction yields accurate results, it is computationally expensive. Parallelization of this algorithm will make it more viable for larger scale datasets.

Related Work

With the explosion of biological data over the last decade, many biological databases such as GenBank, HapMap, PRIDE, and PeptideAtlas make good targets for large scale analysis. Recent studies have used large datasets to examine the origins of new human genes from noncoding DNA [2], the origins of domesticated dogs [3], and the construction of a large scale phylogenetic tree to examine the relationships between eukaryotic groups [4]. The latter study used Maximum Parsimony methods to classify 73,060 taxa between 3 machines in parallel, and took 2.5 months to complete [4]. However, these recent studies had to compromise between input data size and computational time required to remain feasible.

Research involving these large datasets will require fast and efficient algorithms that are capable of handling large volumes of data. Unfortunately, the computational complexity of Maximum Likelihood has previously restricted its viability to smaller datasets due to time constraints. However, because ML is a statistical model where it generates a large amount of candidate trees and evaluates them all for their probability, it could be possible to parallelize the algorithm at certain sections to get a speedup in performance. Potentially speeding up the algorithm could allow for larger datasets to be processed in a reasonable time and yield better results.

Instead of writing our own implementation of the maximum likelihood method [5], we decided to attempt improving the open source Phylip package [6]. The DNAml program generates phylogenetic trees using Maximum Likelihood and was the focus of our parallelization efforts. In order to introduce parallelism to DNAml, we chose to investigate parallelization using GPU programming with CUDA and OpenCL, as well as multithreaded execution via OpenMP.

GPU programming is a relatively new field in parallelization as that graphical hardware is by nature highly parallel. Two of the most prominent languages on this platform are Compute Unified Device Architecture (CUDA) from NVIDIA, and Open Computing Language (OpenCL) from Khronos. Both languages aimed to exploit the highly parallel nature of GPUs and increased the level of parallelism available to programmers. CUDA is a vendor specific language, which has the advantage of direct access and mapping to NVIDIA hardware, while OpenCL is designed to be cross-platform, where kernel code is compiled just in time before executing.

MPI, or the Message Passing Interface, was outside the scope of this paper because an MPI implementation of the DNAml algorithm already exists [7]. As well, there was an optimized version of DNAml called fastDNAml using a custom multithreaded approach [8]. This was also outside the scope of this paper because the multithreading methods employed depended on heavy optimization and had a complex code base.

Proposed Solution

Maximum Likelihood phylogenetic tree reconstruction algorithms are relatively well documented and used. Instead of building our own implementation of ML tree generation, we decided to build on top of the pre-existing DNAml program. By finding ways to parallelize the DNAml program, the hope is to optimally introduce a decrease in time required to generate phylogeny trees, and in turn allow for larger datasets to be processed in a reasonable timeframe.

In order to parallelize the algorithm effectively, we would need to know where the code was spending the largest amount of computation time on. This was done by inserting hooks into the program at the start and end of every function such that every function called by the program could be counted and timed. This information would give insight into which functions did the heavy lifting, their call order, and provide a general area on where to further investigate.

After identifying the key functions and other related computational "hotspots", smaller sections of code would be identified that could be performed in parallel with each other. These identified parallel sections of code were carefully isolated from the rest of the serial code to maintain code independence and avoid potential unintended side effects. Testing was performed on a provided detached node on the Westgrid Checkers cluster with a variety of input files varying the number of sequences and base pairs. The provided detached node had 8 CPU cores, 2 GPUs, and 16GB of memory.

Scripts to generate test sequences as well as execute tests for each version of DNAml were written to streamline the testing. A timing library was also developed to profile the function calls within the codebase. As well, execution timestamps of those functions would be returned in microseconds format. The timing library intercepts the hooks inserted into the codebase and logs the intercepts. Analysis and data representation was done using the statistical software R.

Evaluation

After profiling the DNAml codebase, the two functions evaluate and slopecurv were found to have the most computation time. Slopecurv was called about an order of magnitude higher than the majority of the rest of the functions, while each evaluate call took a significant amount of computation time. As well, it was discovered that much of the time was spent allocating nodes in memory for the tree and rearranging them to generate different tree arrangements. Further examination revealed that under normal circumstances, the code was performed iteratively and could not be broken up without rewriting a significant portion of the algorithm.

In order to allow for some parallelization to work, we targeted the probability categories (probcats) for the sequences, resulting in portions of the code being run for each probability category. There are from 1 to 9 categories are randomly assigned from a gamma distribution [7]. We ran our tests for both 1 and 8 probability categories on a single node from the Checkers Westgrid cluster and immediately discovered that CUDA, OpenMP, and OpenCL drastically reduced the performance of the algorithm.

No attempts at parallelization yielded any speedup, and the OpenCL and CUDA implementations were actually too slow to complete in any reasonable amount of time. We believed that the CUDA implementation slowed down mainly due to the severe increase in memory allocations. To compensate, all allocations were performed only once and those allocations were reused. The time spent in the slopecurv function for CUDA as compared to the original serial version is shown in Figure 1.

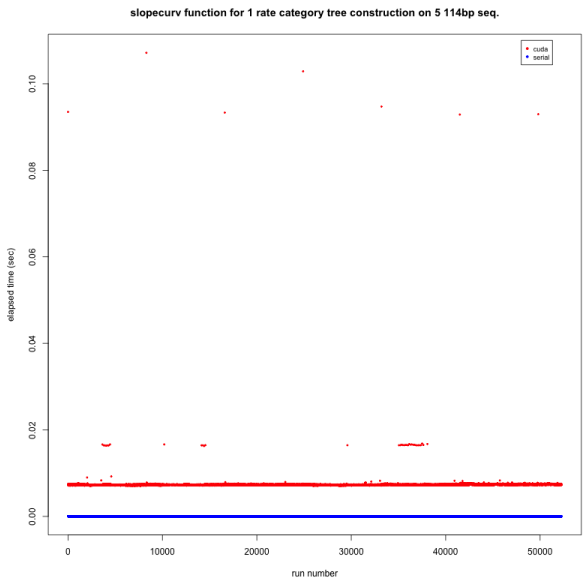

Figure 1. Time spent in slopecurv function for both serial and CUDA implementations of DNAml constructing a tree for 5 sequences of 114 basepairs with one probcat.

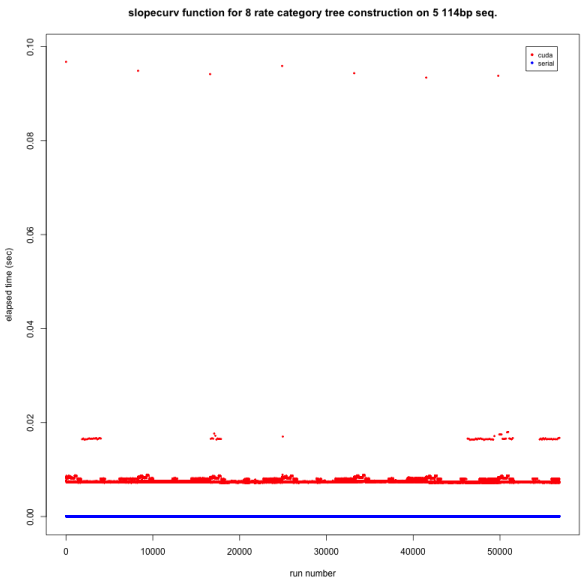

Figure 2. Time spent in slopecurv function for both serial and CUDA implementations of DNAml constructing a tree for 5 sequences of 114 basepairs with 8 probcats.

CUDA was two orders of magnitude slower on average, with some spikes as high as 5 orders of magnitude slower than the serial version. In the one probcat case, CUDA was expected to be slower as there is no parallelization in that instance. However, the overhead introduced from copying data to and from the device was quite significant in this case. Increasing the number of probcats to 8 caused a 3 times slowdown in the serial version, but no significant slowdown in the CUDA version. This implies that the parallel sections introduced are working sufficiently, but the overhead introduced from transferring data to and from the GPU grossly outweighs the benefits.

With probcat at 8, there was approximately 8 times the amount of data being transferred between the GPU and host. However, there was only a small observable speed decrease, implying that the overhead was introduced from the amount of memory copies that were performed, or the act of invoking the kernel and starting threads on the GPU.

OpenMP had a similar trend. In general, the algorithm ran up to 60 times slower than the serial version. This limited the number of tests we used to 100 to gather data as each OpenMP run was consistently much slower than serial. Although this is a slow result, both the CUDA and OpenCL versions of the program were unable to finish within any reasonable amount of time. Unlike the serial version, the time difference between the 1 and 8 probcats were relatively similar. This again suggest that parallelization is working, but the overhead introduced makes parallelization not feasible.

Table 1. Summary of time spent in the slopecurv function for both serial and CUDA implementations of DNAml constructing a tree for 5 sequences of 114 basepairs

	CUDA 1 probcat	serial 1 probcat	CUDA 8 probcat	serial 8 probcat
mean (sec)	0.007361405	1.136362e-05	0.007491643	3.65226e-05
median (sec)	0.007333994	1.096725e-05	0.007380009	3.600121e-05
min (sec)	0.00706315	1.001358e-05	0.007116079	3.480911e-05
max (sec)	0.1071458	2.884865e-05	0.09868288	7.510185e-05

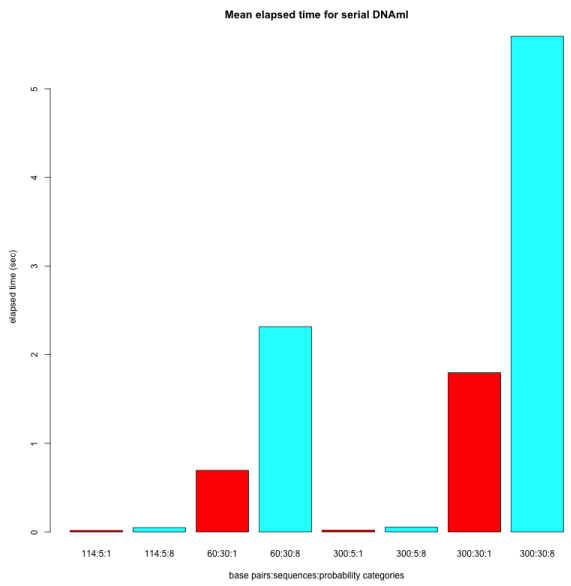

Figure 3. Mean elapsed times for serial runs with varying basepairs, number of sequences, and probability categories.

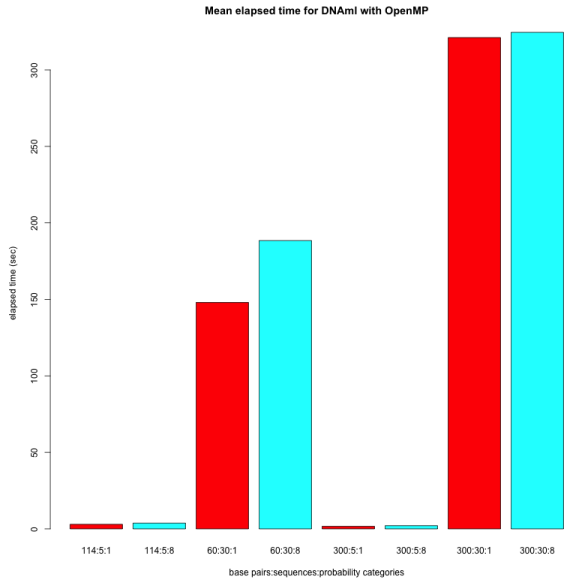

Figure 4. Mean elapsed times for OpenMP runs with varying basepairs, number of sequences, and probability categories.

As seen in Figures 3 and 4, parallelization is working, but the introduced overhead negates that benefit. The overhead could either be from the redundant copying of data for the parallel constructs to use, unshared variables in OpenMP, or the introduction of numerous fork-join constructs such that the actual length of the the parallel section isn't long enough to warrant a fork-join to begin with.

To mitigate some potential data flow bottlenecks, an attempt was made to migrate some of the data structures directly into the GPU using CUDA to reduce the number of host to device copies. This was problematic because the device and host structures had to be synchronized, and copying the memory directly was complex and consisted of many pointers. This forced the memory transfer to come from multiple smaller memory copies from random regions in both GPU and CPU memory instead of one contiguous block. This could be done by rewriting the entire data structure, but any such optimizations would negate any other optimizations in the surrounding code that may rely on that data structure.

Future Work

Given that there are only between 1 and 9 probability categories, its potential for large scale parallelization was severely limited. The real candidate to allow speedup would be the creation of specific trees in parallel. This would allow for many trees to be created and evaluated concurrently, and then the best one chosen out of the set is returned. Unfortunately, using the existing code base would require a massive overhaul of most of the functions. So much in fact that it may justify rewriting the entire algorithm with parallel functionality in mind.

More work with CUDA could have been done by trying to migrate the entire algorithm to the GPU. This would remove the time spent moving data between the GPU and CPU, and would also give the advantage of exploiting the texture cache to store the static data structures and allow for quick access from all GPU threads. As well, the overhead of kernel invocation versus the overhead of copying many small portions of memory into the GPU could be investigated. If the overhead for invoking a kernel is significant enough to cause a bottleneck, then there would exist a minimum time that sequential functions would have to exceed to make parallelization feasible. Such functions could be filtered out during profiling.

Another approach with GPU's could look at MPI-PHYLIP and migrate each MPI rank onto a GPU core

and compare the cost of interprocess communication with the overhead of running on the GPU. In general, this study found that the overhead introduced usually came from data transfer, or a significantly high number of fork-join events. Investigating other methods to reduce the required data transfer or the number of fork-join events should be considered.

Conclusion

Constructing phylogenetic trees with Maximum Likelihood is an accurate but computationally expensive procedure. It is definitely useful to achieve a speedup in DNAml's performance within the Phylip package as that it would enable larger datasets to be feasibly analyzed. However, in order to have the speedup be feasible, it would require a complete rewrite with parallelization as the main focus. As it stands, the current aging codebase does have some sections of potential parallelization. However, in order for true large scale parallelization to occur on DNAml, the entire flow of data and code control has to be rebuilt from the ground up.

Significant bottlenecks encountered in our attempts to parallelize the algorithm included memory allocation, bandwidth between the host and the device, and generally the lack of large parallelizable sections of code. Although we were able to parallelize multiple small sections of code to be parallel, the fork-join model of computation only works well when there are a limited amount of forks and joins. Since the parallelizable sections were not large enough as compared to the rest of the codebase, the majority of the overhead came from the significantly large amount of forks and joins introduced.

As the volume of biological data grows, the demand for algorithms that can handle these datasets becomes more of a necessity. Parallelization can and should be done on many algorithms, but the majority of the aging codebase is not conducive of incremental and progressive parallelization. A ground up design effort is required in order to fully exploit the parallelism that is emerging in the hardware available to researchers.

Works Cited

[1] Fred Opperdoes (1997, August 8). Maximum Likelihood [Online]. Available http://www.icp.ucl.ac.be/~opperd/private/max_likeli.html

[2] Bridgett M. vonHoldt, John P. Pollinger, Kirk E. Lohmueller, et. al., "Genome-wide SNP and haplotype analyses reveal a rich history underlying dog domestication." Nature, vol 464, Apr. 2010.

[3] Dong-Dong Wu, David M. Irwin, Ya-Ping Zhang. "De Novo Origin of Human Protein-Coding Genes." PLoS Genetics, vol. 7, Nov. 2011.

[4] Pablo A. Goloboffa, Santiago A. Catalanob, J. Marcos Mirandeb, Claudia A. Szumika, J. Salvador Ariasa, Mari Kallersjo and James S. Farrisd. "Phylogenetic analysis of 73 060 taxa corroborates major eukaryotic groups." Cladistics. vol 25. pp. 211–230, Sep. 2009.

[5] Joseph Felsenstein. "Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach." Journal of Molecular Evolution, vol. 17, pp. 368-376, 1981

[6] Joseph Felsenstein. "PHYLIP (Phylogeny Inference Package) version 3.5c." Distributed by the author. Department of Genetics, University of Washington, Seattle. 1993.

[7] Alexander J. Ropelewski, Hugh B. Nicholas Jr., Ricardo R. Gonzalez Mendez. "MPI-PHYLIP: Parallelizing Computationally Intensive Phylogenetic Analysis Routines for the Analysis of Large Protein Families." PLoS One, vol. 5, Nov. 2010.

[8] Olsen, G. J., Matsuda, H., Hagstrom, R., and Overbeek, R. "fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood." Computational Applied Bioscience. vol. 10, pp. 41-48. 1994.