Phylogenetics on CUDA (Parallel) Architectures
Bradly Alicea
http://www.msu.edu/~aliceabr
Phylogenetics Introduction

**Q:** How do we understand evolutionary relationships (e.g. common ancestry, descent with modification)?

*use a directed, acyclic graph (from root to last common ancestor to descendents).*
Why Phylogenetics?

The study of evolution is subject to incomplete/non-uniform sampling:

**Fossil record**

**Properties of traits**

**Diversity**

**Answer:** inferential tools and data fusion (consensus between different types of data).

Phylogeny is the hypothetical relationship between closely-related species.

Common ancestor A, how did B and C diverge (black box)?

Potential answer: trace changes one at a time (one possible set of changes).
Computational Complexity

Sampling of taxa (e.g. species, columns in $M_{ij}$) + characters (e.g. DNA sequences, rows in $M_{ij}$) = large # of tree topologies (combinatorics problem).

$$T = \frac{(2n - 5)!}{2^{n-3} (n - 3)!} \quad \text{for } n \geq 3$$

where $n - 1$ is # of taxa (rooted case).

4 taxa, $T = 15$
8 taxa, $T = 135135$
14 taxa, $T \approx 7.906 \times 10^{12}$

Unique perfect phylogeny (best evolutionary hypothesis) is NP-hard (for $M_{ij} \geq 10 - 11^2$), so we use heuristic techniques to approximate answer.

1) Maximum parsimony: nonparametric, best tree is one with least number of unique changes. Minimax method.

2) Maximum likelihood: parametric model estimated from data. Probability distribution for particular tree topologies.

<table>
<thead>
<tr>
<th>$M_{ij}$</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
<th>$T_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>ATA</td>
<td>ATA</td>
<td>ATA</td>
<td>ATC</td>
<td>ATC</td>
</tr>
<tr>
<td>$C_2$</td>
<td>TTG</td>
<td>TTC</td>
<td>TTG</td>
<td>TTG</td>
<td>TTT</td>
</tr>
<tr>
<td>$C_3$</td>
<td>TAG</td>
<td>TAG</td>
<td>TAG</td>
<td>TAG</td>
<td>TAG</td>
</tr>
<tr>
<td>$C_4$</td>
<td>CAC</td>
<td>CAC</td>
<td>CAA</td>
<td>CAA</td>
<td>CAA</td>
</tr>
<tr>
<td>$C_5$</td>
<td>AAG</td>
<td>AAG</td>
<td>AAA</td>
<td>AAA</td>
<td>AAC</td>
</tr>
</tbody>
</table>
Four-taxon problem: toy problem for comparing parsimony and likelihood methods.

1. What methods recover the true tree most often?

Maximum Parsimony: gradient descent, lowest scores win (generates “best” trees).

Maximum Likelihood: search space pruned using dynamic programming, generates series of “best” subtrees.

2. What methods are better at exception-handling?

1) Long-branch attraction: when two branches are reported to have same common ancestor but should be phylogenetically distant.

2) Polytomy: when many branches occur at the same time (rapid speciation, lack of resolution).

1. What methods recover the true tree most often?

Maximum Parsimony: gradient descent, lowest scores win (generates “best” trees).

Maximum Likelihood: search space pruned using dynamic programming, generates series of “best” subtrees.

2. What methods are better at exception-handling?

1) Long-branch attraction: when two branches are reported to have same common ancestor but should be phylogenetically distant.

2) Polytomy: when many branches occur at the same time (rapid speciation, lack of resolution).
Example #1

**Suchard and Rambaut, Many-core algorithms for statistical phylogenetics. Bioinformatics, 25(11), 1370-1376.**

Bayesian model (e.g. Maximum Likelihood):
* 62 mitochondrial genomes, carnivores (60 state codon model).

* 90-fold increase over CPU models.


Nucleotide-based model: each base (A, C, G, T) serves as a character.
* 4 possible states per character, total # characters = total # of nucleotide positions sampled (n).

Codon-based model: each combination of three nucleotides (e.g. ACT, TCT, GAG) serves as a character.
* 64 possible states ($4^3$) per character, total # characters = n / 3.
Model and Approach to Likelihood

\[ F_u = \{ F_{urcs} \} \rightarrow \text{matrix of size } R \times C \times S \]

* element \( F_{us} \) = probability that node \( u \) is in state \( s \).

Travel upward through tree, compute forward likelihoods for each internal node \( u \).

When data are unambigious, algorithm runs in \( O(RCS^2) \) time.

Algorithm 1: GPU-based parallel computation of partial-likelihoods

1. Define COLUMN_BLOCK_SIZE (CBS) \( \leftarrow \) number of columns processed per thread block
2. Define STATE_BLOCK_SIZE (SBS) \( \leftarrow \) number of states processed per inner-loop
3. for all thread blocks (rate class \( r = 1, \ldots, R \) and column-block \( c = 1, \ldots, [C/CBS] \) in parallel do
4. using each thread \( t = 1, \ldots, S \) in SBS block, prefetch child partial likelihoods \( F_{urcs} \) for CBS columns (reused by all threads in block)
5. Initialize \( F_{urcs}^{(1)} \leftarrow 0 \) and \( F_{urcs}^{(2)} \leftarrow 0 \)
6. for \( j = 1 \) to S in SBS increments do
7. Pre-fetch transition probabilities \( P_{ij}(r)(t_b) \) for SBS states (reused by all threads in block)
8. \( F_{urcs}^{(1)} \leftarrow F_{urcs}^{(1)} \times P_{ij}(r)(t_b) \)
9. \( F_{urcs}^{(2)} \leftarrow F_{urcs}^{(2)} \times P_{ij}(r)(t_b) \)
10. end for
11. end for
12. Return \( F_{urcs}^{(1)} \times F_{urcs}^{(2)} \)
Many-core Implementation

Many-core implementation:
* for each node $u$ recursively (every $u, c, s$) – distribute so that each $(r; c, s)$ entry executes its own thread.

* for each $(r; c, s)$, small portion of code dedicated to computing $F_{ures}$.

* previous approach to parallelization: partition columns into conditionally-independent blocks, distribute blocks across separate cores.

Four vectors calculated:
1-2) Reading forward likelihoods of child nodes $\{F_{\phi(bn)rej}\}$.

3-4) Finite-time transition probability matrices $\{P_{sj}(t|t_{bn})\}$.

GPU coalesces memory read/writes of 16 consecutive threads into single transaction.

Zero-padding used to deal with stop codons, other zero-probability states.
CPU vs. GPU

MCMC (Markov Chain Monte Carlo)-based resampling techniques implemented in single (32-bit) and double (64-bit) precision.

* in both cases, large speedup vs. CPU.

Important scaling dimensions:

1) state-space size (S). For nucleotides (S= 4), speedup using GPU becomes more modest.

2) alignment columns (C) – very small, large C values show limited benefit on GPU.

3) number of taxa (N) – performance saturates on GPU for larger values.
Reconstructed Tree, GPU

Tree reconstructions form “clades”, or sets of related taxa.

Clade membership = statistical support:
* maximum parsimony: bootstrap values.
* maximum likelihood: posterior probabilities.

Outgroup: helps to order the tree topology (distantly-related species).

Molecular clock: hypothesized rate of change for specific gene (can be a constant).
Example #2


Implemented MrBayes, a commonly-used program for phylogenomics, on multi-core CPU and GPGPU.

* fine-grained parallelism = very small threads.

* compute phylogenetic likelihood function (PLF) on known phylogeny, ultimate goal is benchmarking rather than reconstruction.

* can GPU improve inferential power of ML model?

Goal is to speed-up computation of the PLF in parallel, so that in application search time among a large amount of potential tree topologies can be minimized.
Rooting/Likelihood Construction

MrBayes (Bayesian inference), Maximum Likelihood.

Phylogenetic Likelihood Functions (PLF) require substitution, conditional probability information.

* estimate branch lengths and parameters of nucleotide substitution model (matrix Q, Figure 2).

* three main functions: CondLikeDown, CondLikeRoot, CondLikeScaler.

PLF = independent for loops, load depends on sequence length (m) and number of discrete rates (r).

* Down, Root - multiply likelihood vector elements by substitution matrix for each discrete rate (16 inner products required). Scaler - properly scale computation.
Partitioning Data for GPU

Phylogenomic data example: 1500 genes, ML analysis = 2+E06.

Efficient use of the GPU:
Number of threads must be maximized using one of two approaches:

1) direct parallelization (use group of threads in parallel, requires large number of synchronization points and conditional statements).

2) parallelize work at the likelihood vector entry level.

* calculation of each vector entry assigned to independent threads (avoid overhead).

Second approach: less concurrency and completely independent threads.
Final Issue: Scalability

Scalability: input data characterized by leaves (taxa) and columns (characters):
* leaves (number of calls to the PLF), columns (size of data computed in loops).

* multi-core CPU architectures: efficiency of fine-grained parallelism depends on number of cores sharing resources (intra-chip, cross-core communication).

* GPU: speedup as dataset increases (up to 20-50K columns). Speedup increases with computational intensity (GPU optimized for executing small parallel threads). GPU benefit realized when computation-to-data ratio is high.

GPU vs. CPU: neither method is uniformly superior:
* largest overhead for GPU: transfer of data (overall, need for better data sharing and more efficient communication mechanisms).

* parallelization time (programming) for MrBayes: 1/2 day for multi-core CPU vs. 2 days on GPU. CPU wins on this count.

* both multi-core CPU and GPU reduce amount of time consumed in the execution of the PLF. For baseline, 90%. Multi-core CPU = 10-15% vs. GPU = 5-10%. GPU wins on this count.