Highly Scalable Genome Assembly on Campus Grids

Christopher Moretti
Michael Olson, Scott Emrich, Douglas Thain

University of Notre Dame
Overview

- Scientists get stuck in a loop: CODE → DEBUG → SCALE UP → RE-CODE ...

- We believe:
  - The Many-Task Paradigm: coordinating 1000s of serial programs on commodity hardware is an effective mechanism for designing solutions that don’t require scientists to change their existing solutions when scaling up to multi-institutional campus grid resources.
Genome Assembly

Genome sequencing extracts DNA \{A,G,T,C\} from biological samples in reads of 25-1000 bases each.

Biologists need much longer DNA strings to perform their analyses.

Assembly is the process of putting the pieces together into long contiguous sequences.
Assembly Pipeline

(1) Unordered reads from sequencing
Assembly Pipeline – Candidate Selection

(2) Candidates based on short exact matches
Assembly Pipeline – Alignment

(3) Actual Overlaps are Computed
(4) Alignments are ordered and combined into contigs
**Complete Assembly of A. Gambiae Mosquito**

<table>
<thead>
<tr>
<th></th>
<th>Candidate Sel.</th>
<th>Alignment</th>
<th>Consensus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celera</td>
<td>Combined</td>
<td>4.5 hours</td>
<td>3 hours</td>
</tr>
<tr>
<td>Complete SW</td>
<td>(1.5 hrs serially)</td>
<td>(12 days serially)</td>
<td>3 hours</td>
</tr>
<tr>
<td></td>
<td>5 minutes</td>
<td>45 minutes</td>
<td></td>
</tr>
<tr>
<td>Banded SW</td>
<td>(1.5 hrs serially)</td>
<td>(7 hrs serially)</td>
<td>3 hours</td>
</tr>
<tr>
<td></td>
<td>5 minutes</td>
<td>11 minutes</td>
<td></td>
</tr>
</tbody>
</table>

Similarly, we can bring the candidate selection and alignment time for the much larger *S. Bicolor* grass down from more than 9 days on Celera to 3 hours (Complete) and 1.25 hours (Banded).

So why did we choose to attack Candidate Selection and Alignment? And what about Amdahl’s Law?
Candidate Selection

1M reads → 1 trillion alignments
8M reads → 64 trillion alignments
... 50,000 CPUYears!

\( k \)-mer counting heuristic:

“two sequences that share a short exact match are more likely to overlap significantly than two sequences that don’t share an exact match”

Even optimized \( k \)-mer counting is extremely memory intensive – 16GB for the 8M read data set. Worse, it is not naturally parallelizable.
Parallel Candidate Selection

We chose to trade off increased computational complexity for the ability to parallelize the Candidate Selection with 10,000’s of separate tasks and decreased memory consumption per node.

\(k\)-mer counting is \(O(nm)\) – \(n\) reads of average length \(m\)

Instead, we divide the input into \(n/l\) subsets of size \(l\)

Compute every pair – \(O(n^2/l^2)\) – each completed in \(O(lm)\)

For a total complexity of: \(O(n^2m/l)\)
Parallel Candidate Selection

We chose to trade off increased computational complexity for the ability to parallelize the Candidate Selection with 10,000’s of separate tasks and decreased memory consumption per node.

$k$-mer counting is $O(nm) – n$ reads of average length $m$

Instead, we divide the input into $n/l$ subsets of size $l$

Compute every pair – $O(n^2/l^2)$ – each completed in $O(lm)$

For a total complexity of: $O(n^2m/l)$
Alignment

Now we have candidate pairs whose alignment can be computed independently in parallel using sequential programs:

```bash
for i in Candidates; do
  batch_submit aligner $i
done
```

What’s wrong with this?

- Batch system latency
- Local and remote replication of many copies of each sequence and/or requirement of a global FS
Candidate (Work) List

Seq1  Seq2
Seq1  Seq3
Seq2  Seq3
Seq4  Seq5

put “Align”
put “>Seq1
ATGCTAG
…” 1.in
run “Align < 1.in > 1.out”
get 1.out

Worker

Master

Input Sequence Data

Output Alignment Results
(raw format)
12.6M candidates from 1.8M reads.
121.3M candidates from 7.9M reads.
Scaling to larger numbers of workers
Scaling to larger numbers of workers

\[ W \text{ (idle)} \rightarrow M \rightarrow \ldots \rightarrow W \text{ (idle)} \]
Scaling to larger numbers of workers

W (busy)  W (idle)  W (idle)  ...  W (idle)  W (idle)
Scaling to larger numbers of workers

W (busy)  W (busy)  W (idle)  ...  W (idle)  W (idle)
Scaling to larger numbers of workers

\[ \text{M} \]

\[ \text{W (busy)} \quad \text{W (busy)} \quad \text{W (busy)} \quad \ldots \quad \text{W (idle)} \quad \text{W (idle)} \]
Scaling to larger numbers of workers

This is exacerbated when network links slow down, for instance when harnessing resources at another institution.
Putting it all together

Finally, we can run our distributed Candidate Selection and Alignment concurrently in order to pipeline these stages of the assembly (and save a bit of time versus the two modules run back-to-back).

Inserting our distributed modules in place of the default candidate selection and alignment procedures, we decrease these two steps of the assembly **from hours to minutes** on one of our genomes, and **from nine days to less than one hour** on our largest genome.
For More Information

- Christopher Moretti and Prof. Douglas Thain
  - Cooperative Computing Lab
    - http://cse.nd.edu/~ccl
    - cmoretti@cse.nd.edu
    - dthain@cse.nd.edu

- Michael Olson and Prof. Scott Emrich
  - ND Bioinformatics Laboratory
    - http://www.nd.edu/~biocmp
    - molson3@nd.edu
    - semrich@nd.edu

- Funding acknowledgements:
  - University of Notre Dame strategic initiative for Global Health.
  - National Institutes of Health (NIAID contract 266200400039C)
  - National Science Foundation (grant CNS06-43229).
### Time to Assemble *A. Gambiae*

<table>
<thead>
<tr>
<th>Assembly Framework</th>
<th>Alignment Algorithm</th>
<th>System Size</th>
<th>Candidate Selection</th>
<th>Alignment</th>
<th>Consensus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celera</td>
<td>Celera</td>
<td>4 cores</td>
<td></td>
<td>4 hrs, 20 min</td>
<td>3 hrs, 11 min</td>
</tr>
<tr>
<td>Modular</td>
<td>Complete</td>
<td>1 core</td>
<td>* 1 hr, 35 min</td>
<td>* 12 days, 4 hrs</td>
<td></td>
</tr>
<tr>
<td>Modular</td>
<td>Banded</td>
<td>1 core</td>
<td>* 1 hr, 35 min</td>
<td>* 7 hrs, 9 min</td>
<td>2 hrs, 33 min</td>
</tr>
<tr>
<td>Modular</td>
<td>Complete</td>
<td>campus grid</td>
<td>5 min</td>
<td>45 min</td>
<td></td>
</tr>
<tr>
<td>Modular</td>
<td>Banded</td>
<td>campus grid</td>
<td>5 min</td>
<td>11 min</td>
<td></td>
</tr>
</tbody>
</table>

### Time to Assemble *S. Bicolor*

<table>
<thead>
<tr>
<th>Assembly Framework</th>
<th>Alignment Algorithm</th>
<th>System Size</th>
<th>Candidate Selection</th>
<th>Alignment</th>
<th>Consensus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celera</td>
<td>Celera</td>
<td>4 cores</td>
<td>crashed after 9 days</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>Modular</td>
<td>Complete</td>
<td>1 core</td>
<td>* 14 hr, 39 min</td>
<td>* 50 days, 15 hrs</td>
<td>17 hrs</td>
</tr>
<tr>
<td>Modular</td>
<td>Banded</td>
<td>1 core</td>
<td>* 14 hr, 39 min</td>
<td>* 46 hrs, 17 min</td>
<td></td>
</tr>
<tr>
<td>Modular</td>
<td>Complete</td>
<td>campus grid</td>
<td>34 min</td>
<td>2 hrs, 40 min</td>
<td></td>
</tr>
<tr>
<td>Modular</td>
<td>Banded</td>
<td>campus grid</td>
<td>34 min</td>
<td>43 min</td>
<td></td>
</tr>
</tbody>
</table>
How?

- **On my workstation.**
  - Write my program, make sure to make it partitionable, because it takes a really long time and might crash, debug it. Now run it for 39 days – 2.3 years.

- **On my department’s 128-node research cluster**
  - Learn MPI, determine how I want to move many GBs of data around, re-write my program and re-debug, wait until the cluster can give me 8-128 homogeneous nodes at once, or go buy my own. Now run it.

- **BlueGene**
  - Get $$$ or access, learn custom MPI-like computation and communication working language, determine how I want to handle communication and data movement, re-write my program, wait for configuration or access, re-debug my program, re-run.
So?

- Serially
- Cluster
- Supercomputer

So I can either take my program as-is and it’ll take forever, or I can do a new custom implementation to a certain particular architecture and re-write and re-debug it every time we upgrade (assuming I’m lucky enough to have a BlueGene in the first place)?

- Well what about Condor?