Large-scale parallel computing for computational genomics

Aydın Buluç
Computational Research Division, LBNL
EECS Department, UC Berkeley

H.Bioinfo-11, Thessaloniki
November 17, 2018
Microbiome Analysis is a Science & Health Problem

Health

Bio-Energy

Environment

Bio-Manufacturing

Biology Foundations
# ExaBiome: Exascale Solutions to Microbiome Analysis

Exascale algorithms & systems for previously intractable problems

<table>
<thead>
<tr>
<th>Problem Domain</th>
<th>Metagenome Assembly</th>
<th>Protein Clustering</th>
<th>Comparative Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Science need</td>
<td>Discover, understand (find genes) and control species in microbial communities</td>
<td>Improve understanding of tree of life for microbes; aid in identifying gene function</td>
<td>Track microbiome over time, different behavior due to changes in to environment, climate, drugs, food, species, etc.</td>
</tr>
<tr>
<td>Computing techniques</td>
<td>Hash tables, graphs, alignment</td>
<td>Clustering (ML), graphs, sparse linear algebra</td>
<td>Alignment, dimensionality reduction (ML), hashing</td>
</tr>
</tbody>
</table>

In this talk, I will briefly cover the first two pillars.
The metagenome assembly problem

Environmental sample (e.g. bacterial communities)

Sequencer

Short "reads" with errors (100-300 bases long)

Underlying genomes (up to $10^7$ bases long each)
De novo genome assembly is hard !!!

- There is no genome reference.
  - In principle we want to reconstruct unknown genome sequences.

- Reads are significantly shorter than underlying genome.
  - Reads ~200 bases long VS bacterial genomes ~10^7 bases long

- Reads include errors.
  - On Illumina reads the error rate is 0.1%
De novo metagenome assembly is harder !!!

- Variable frequency (abundancy) of the genomes within the sample
- Repeated sequences across genomes
- Polymorphism within species (“similar but not identical”)
Jigsaw puzzle analogy

X 10  X 5  X 1
Metagenome assembly is expensive !!!

- Expensive in both memory requirements and runtime

- State-of-the-art tools can only run on single node
  - Can handle only small/medium datasets
  - Subsampling to fit node → compromises quality
  - Quality tuning is not viable (e.g. parameter sweeps)

- Latest metaSPAdes (3.13.0) on biofuel (117 GB, modest size)
  - 2 days and 403 GB of memory on 80 core machine !!!
MetaHipMer assembly pipeline

• MetaHipMer is the first massively scalable & high-quality metagenome assembly pipeline.

• The **first distributed-memory metagenome assembler** that achieves comparable quality to state-of-the-art tools.

• Scales to tens of thousands of cores
  → decreases execution time by orders of magnitude

• The pipeline consists of two components:
  1. The iterative contig generation algorithm
  2. Scaffolding algorithm
Iterative contig generation algorithm

1. k-mer analysis
2. de Bruijn graph traversal
3. bubble merging & hair removal
4. iterative graph pruning
5. reads to contigs alignment
6. local assembly
7. extract (k+s)-mers from contigs
8. iterate for k=k+s
The MetaHipMer scaffolding algorithm

- Read to contig alignment
- Contig link generation
- Contig graph traversal
- Gap closing
- Final scaffolds
MetaHipmer’s fundamental data structure is a distributed hash table

- The k-mer analysis / contig indexing use hash tables. **WHY?**
  - Direct index for the k-mers is not practical for realistic values of $k$
    $\Rightarrow 4^k$ different k-mers!

- The de Bruijn graph of k-mers is represented as a hash table. **WHY?**
  - Adjacency matrix of a huge de Bruijn graph (soil metagenome) is an extremely sparse $10^{11} \times 10^{11}$ matrix with 2 to 8 non-zeros per row!
Why *distributed* hash tables?

- Hash tables representing the de Bruijn graphs are huge
  - 100s of GBs up to 100s of TBs

*Aggregate distributed memory for memory requirements*

- Want to parallelize the underlying algorithms for speed
  - Scaling-out irregular algorithms is challenging

*Distributed & globally accessible hash tables*
Parallelization strategy

• The involved algorithms are inherently irregular
  → irregular & all-to-all communication patterns

• Core ideas for efficient parallelization

  1. Design parallel algorithms using a PGAS paradigm
  2. Use distributed hash tables
  3. Optimize common use-cases of hash tables
  4. Understand data/bottlenecks and iterate 1, 2, 3

PGAS: Partitioned Global Address Space
Distributed hash table in a PGAS model

- **PGAS:** Partitioned **G**lobal **A**ddress **S**pace
  - Bucket/entries of hash table allocated in shared address space
- **U**nified **P**arallel **C** (UPC) is a PGAS parallel language
  - Threads can access any other bucket/entry with read/write instructions
  - One sided communication facilitates irregular accesses

![Partitioned Global Address Space](image)

- Distributed memory
MetaHipMer produces high-quality results

<table>
<thead>
<tr>
<th>Assembler</th>
<th>misassemblies</th>
<th>rRNAs count</th>
<th>Length in (MBp) pieces &gt;50k</th>
<th>Genome fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaHipMer</td>
<td>682</td>
<td>79</td>
<td>108</td>
<td>94%</td>
</tr>
<tr>
<td>MetaSPAdes</td>
<td>914</td>
<td>50</td>
<td>115</td>
<td>94%</td>
</tr>
<tr>
<td>Megahit</td>
<td>761</td>
<td>68</td>
<td>103</td>
<td>95%</td>
</tr>
<tr>
<td>Ray Meta</td>
<td>793</td>
<td>75</td>
<td>79</td>
<td>88%</td>
</tr>
</tbody>
</table>

- Comparative assembly quality results on MG64
  - Synthetic dataset comprising a mixture of 64 diverse bacteria

- This is a small dataset, used just for quality comparison
  - We have the reference sequences
MetaHipMer strong scaling

- Run on a subset of the Wetlands (soil metagenome) dataset
  - Real, high-complexity dataset
- Experimental platform: a XC40 Cray system (Cori @ NERSC)
- **Strong scaling efficiency 61% from 1K to 32K cores !!!**
Grand challenge problem:  
**Assembly of the full Wetlands dataset**

- Full Wetlands dataset:
  - Massive-scale dataset, consisting of 2.6 Tbytes of raw reads.

- Assembling such datasets with state-of-the-art tools is **intractable**
  - shared memory tools, hence limited available memory
  - previous approaches assembled **subsampled datasets**
  - compromised quality

- MetaHipMer on 512 nodes assembled FULL dataset in 3h25mins

To the best of our knowledge this is the **largest, high-quality de novo metagenome assembly completed to date**.

---

Grand challenge problem:

**Assembly of the full Wetlands dataset**

3-lanes of Wetlands

- 3-lane assembly: 2.3 Gbase pairs
- 7.6% completeness

All 21-lanes of Wetlands

- 21-lane assembly: 41.5 Gbase pairs
- 7x larger assembly
- 42% completeness
Philosophy of the Markov Cluster Algorithm (MCL)

The number of **edges or higher-length paths** between two arbitrary nodes in a cluster is greater than the number of paths between nodes from different clusters.

**Random walks** on the graph will frequently remain within a cluster.

The algorithm **computes the probability** of random walks through the graph and **removes lower probability terms** to form clusters.
Markov Cluster Algorithm (MCL)

Widely popular and successful algorithm for discovering clusters in protein interaction and protein similarity networks.

At each iteration:

**Step 1 (Expansion):** Squaring the matrix while pruning (a) small entries, (b) denser columns

**Naïve implementation:** sparse matrix-matrix product (SpGEMM), followed by column-wise top-K selection and column-wise pruning

**Step 2 (Inflation):** taking powers entry-wise
- **b**: number of columns in the output constructed at once
  - Smaller b: less parallelism, memory efficient (b=1 is equivalent to sparse matrix-sparse vector multiplication used in MCL)
  - Larger b: more parallelism, memory intensive
Combined expansion and pruning

- **b**: number of columns in the output constructed at once
  - HipMCL selects $b$ dynamically as permitted by the available memory
  - The algorithm works in $h = N/b$ phases where $N$ is the number of columns (vertices in the network) in the matrix
HipMCL: High-performance MCL

- MCL process is both **computationally expensive** and **memory hungry**, limiting the sizes of networks that can be clustered.
- HipMCL overcomes such limitation via **sparse parallel algorithms**.
- **Up to 1000X times faster** than original MCL with same accuracy.

\[
\begin{align*}
\sqrt{p} \times \sqrt{p} & \quad \text{Process Grid} \\
A & \times \quad A (\text{or} \ A_b) \\
A^2 & =
\end{align*}
\]

### HipMCL on large networks

<table>
<thead>
<tr>
<th>Data</th>
<th>Proteins</th>
<th>Edges</th>
<th>#Clusters</th>
<th>HipMCL time</th>
<th>platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolate-1</td>
<td>47M</td>
<td>7 B</td>
<td>1.6M</td>
<td>1 hr</td>
<td>1024 nodes Edison</td>
</tr>
<tr>
<td>Isolate-2</td>
<td>69M</td>
<td>12 B</td>
<td>3.4M</td>
<td>1.66 hr</td>
<td>1024 nodes Edison</td>
</tr>
<tr>
<td>Isolate-3</td>
<td>70M</td>
<td>68 B</td>
<td>2.9M</td>
<td>2.41 hr</td>
<td>2048 nodes Cori KNL</td>
</tr>
<tr>
<td>MetaClust50</td>
<td>282M</td>
<td>37B</td>
<td>41.5M</td>
<td>3.23 hr</td>
<td>2048 nodes Cori KNL</td>
</tr>
</tbody>
</table>

**MCL can not cluster these networks**
Conclusions

• **ExaBiome** project uses large-scale parallel computing to address grand-challenge problems in Microbiome analysis.

• **MetaHipMer** is the first massively scalable, high-quality metagenome assembly pipeline.

• **MetaHipMer is transformative**: Previously intractable problems can be now solved with MetaHipmer on a modest amount of nodes within a few hours!

• **HipMCL** can cluster networks orders of magnitude larger than those that can be clustered with MCL, enabling unprecedented discoveries in network biology.

• These capabilities, made possible using **large-scale parallel computing**, will open up a new era in metagenomic analysis.
New Exciting Work in Infancy

- **Long reads** from PacBio and Oxford Nanopore have the potential to revolutionize de-novo assembly.
- **Overlap-Consensus-Layout** paradigm is more suitable than de Bruijn graph paradigm.
- **Overlapping** is the most computationally expensive step.

Read overlapping using shared k-mers is computing a **sparse matrix product**, for which we know good algorithms and implementations.
BELLA: Berkeley Long-read to Long-read Aligner and Overlapper

Number of states: $k + 1$

Legend:

- **State**: correct bases on read, and read$_i$
  - $n$
  - $p^2$
  - $(1 - p^2)$

Preliminary results:

<table>
<thead>
<tr>
<th>Species</th>
<th>BELLA sensitive mode</th>
<th>BELLA precise mode</th>
<th>BLASR</th>
<th>Minimap</th>
<th>Minimap2</th>
<th>MHAP</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>A. baumannii</em></td>
<td>99.58</td>
<td>99.47</td>
<td>89.77</td>
<td>79.69</td>
<td>96.90</td>
<td>76.95</td>
</tr>
<tr>
<td></td>
<td>56.45</td>
<td>83.92</td>
<td>84.56</td>
<td>83.96</td>
<td>71.63</td>
<td>28.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>127.11</td>
<td>11.67</td>
<td>9.42</td>
<td>131.64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><em>C. elegans</em></td>
<td>98.63</td>
<td>97.70</td>
<td>95.75</td>
<td>81.40</td>
<td>95.77</td>
<td>82.64</td>
</tr>
<tr>
<td>20X</td>
<td>47.05</td>
<td>86.88</td>
<td>78.29</td>
<td>78.15</td>
<td>70.62</td>
<td>6.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1,636.95</td>
<td>351.54</td>
<td>209.21</td>
<td>6,218.39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

BELLA addresses:

- How to choose the right set of k-mers, otherwise there are too many of them?
- How to use alignment score to tell true alignments from false positives?

Guidi G, Ellis M, Rokhsar D, Yelick K, Buluç A. BELLA: Berkeley Efficient Long-Read to Long-Read Aligner and Overlapper. bioRxiv 464420; doi: https://doi.org/10.1101/464420
Acknowledgements

Kathy Yelick    Lenny Oliker    Dan Rokhsar    Evangelos Georganas    Nikos Kyrpides    Christos Ouzounis

Steve Hofmeyr    Rob Egan    Georgios Pavlopoulos    Ariful Azad    Giulia Guidi    Eugene Goltsman

Marquita Ellis    Brandon Cook    Bill Arndt    Andrew Tritt