ExaBiome: Exascale Solutions to Microbiome Analysis

https://sites.google.com/lbl.gov/exabiome/
Kathy Yelick (PI)
LBNL, LANL, and JGI

January 2017
The Team from LBNL, LANL and JGI

Kathy Yelick         Lenny Oliker
Dan Rokhsar          Aydin Buluc
Patrick Chain        Nikos Kyrpides
Steve Hofmeyr        Rob Egan
Natalia Ivanova      Ariful Azad
Pavel Senin          Evangelos Georganas
ExaBiome: Exascale Solutions for Microbiome Analysis

- **Microbes**: single cell organisms, such as bacteria and viruses
- **Microbiomes**: communities of 1000s of microbial species, less than 1% individually culturable in a lab (and thus sequenced)
- **Metagenomics**: genome sequencing on these communities (growing exponentially)

Environment  (Health)  Bio-Energy  Bio-Manufacturing
Metagenome database growth

- Metagenomic data double every 11 months
Metagenomics data mining efforts at JGI

Assembled metagenomes

Blind Spots
Science, 2014

Kryptonia
Nature Com., 2016

Recoding
Science, 2014

Earth Virome

Selenocysteine Recoding
A. Chemie, 2016

With D. Soll, Yale.

Biosynthetic clusters
in progress

Novel Protein folds
in preparation

With D. Baker, UW

Metagenomic Protein clusters
in progress
## ExaBiome: Exascale Solutions for Microbiome Analysis

### Metagenome Assembly
- Graph algorithms, Hash Tables, alignment (Smith-Waterman)

### Protein Clustering
- Machine learning (clustering), sparse linear algebra / graphs

### Comparative Analysis
- Alignment, Machine learning (dimensionality reduction), linear algebra

<table>
<thead>
<tr>
<th>Metagenome Assembly</th>
<th>Protein Clustering</th>
<th>Comparative Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph algorithms, Hash Tables, alignment (Smith-Waterman)</td>
<td>Machine learning (clustering), sparse linear algebra / graphs</td>
<td>Alignment, Machine learning (dimensionality reduction), linear algebra</td>
</tr>
<tr>
<td>Fine-grained comm., all-to-all, remote atomics and fast I/O</td>
<td>Fast barriers, subset reductions</td>
<td>All-to-all</td>
</tr>
</tbody>
</table>
Metagenome assembly is hard

• There is no genome reference.
• Reads are significantly shorter than whole genome.
  – Reads consist of 20 to 30K bases
  – Genomes vary in length and complexity – up to 30G bases
• Reads include errors.
• Genomes have repetitive regions.
  – Repetitive regions increase genome complexity
• Microbial genomes occur with thousands of others
  – The coverage (frequency) may vary be orders of magnitude
Strong scaling (human genome) on Cray XC30

Number of Cores vs. Seconds

- Ideal IO cached
- Overall No IO cached
- Overall IO cached
- Kmer analysis
- Contig generation
- Scaffolding
- IO

Seconds:
- 8192
- 4096
- 2048
- 1024
- 512
- 256
- 128
- 64
- 32
- 16

Number of Cores:
- 1920
- 3840
- 7680
- 15360
- 23040
Single Node Scaling on KNL

Strong Scaling (small dataset, chromosome14)

UPC (with MPI in K-mer analysis)
• Complete assembly of human genome in **4 minutes using 23K cores.**
• **700x speedup over** original Meraculous code used in production (only ran on shared memory where it took **2,880 minutes**)
Strong scaling (wheat genome) on Cray XC30

Early version of HipMer were used for the first whole genome assembly of wheat.
**Original HipMer Pipeline Summary (Single Genome)**

**Input:** Reads that may contain errors

Chop reads into k-mers, process k-mers to **exclude errors**

Construct & traverse de Bruijn graph of k-mers, generate **contigs**

Leverage read information to link contigs and generate **scaffolds**.

*(Assembly output is generally called “contigs” even when a scaffolding phase is included.)*
K-Mer Analysis: Pass 1 (I/O + Independent + Reduce)

Reads

Parse to k-mers

k-mer cardinality estimation

Parallel Reduction

Create local Bloom filters

Hyperloglog

Bloom Filter of size \(G/n\)

Bloom Filter of size \(G/n\)

Bloom Filter of size \(G/n\)

\(P_0\)

\(P_1\)

\(P_n\)

Also identify high-frequency k-mers at this step

G = Global cardinality estimation

Also identify high-frequency k-mers at this step
K-Mer Analysis: Pass 2 (Iterative All-to-All)

Reads

<table>
<thead>
<tr>
<th>Parse to k-mers</th>
<th>Hash k-mers &amp; find owners</th>
<th>All-to-all communication of k-mers</th>
<th>Store a k-mer in the local set only if it was seen before</th>
<th>The local set does the actual counting of the k-mers</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>Hash k-mers</td>
<td>Received k-mers</td>
<td>Bloom Filter</td>
<td>Local set</td>
</tr>
<tr>
<td>P₁</td>
<td>Hash k-mers</td>
<td>Received k-mers</td>
<td>Bloom Filter</td>
<td>Local set</td>
</tr>
<tr>
<td>...</td>
<td>Hash k-mers</td>
<td>Received k-mers</td>
<td>Bloom Filter</td>
<td>Local set</td>
</tr>
<tr>
<td>Pₙ</td>
<td>Hash k-mers</td>
<td>Received k-mers</td>
<td>Bloom Filter</td>
<td>Local set</td>
</tr>
</tbody>
</table>

K-Mer Analysis: Pass 2 (Iterative All-to-All)

Reads

<table>
<thead>
<tr>
<th>Parse to k-mers</th>
<th>Hash k-mers &amp; find owners</th>
<th>All-to-all communication of k-mers</th>
<th>Store a k-mer in the local set only if it was seen before</th>
<th>The local set does the actual counting of the k-mers</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>Hash k-mers</td>
<td>Received k-mers</td>
<td>Bloom Filter</td>
<td>Local set</td>
</tr>
<tr>
<td>P₁</td>
<td>Hash k-mers</td>
<td>Received k-mers</td>
<td>Bloom Filter</td>
<td>Local set</td>
</tr>
<tr>
<td>...</td>
<td>Hash k-mers</td>
<td>Received k-mers</td>
<td>Bloom Filter</td>
<td>Local set</td>
</tr>
<tr>
<td>Pₙ</td>
<td>Hash k-mers</td>
<td>Received k-mers</td>
<td>Bloom Filter</td>
<td>Local set</td>
</tr>
</tbody>
</table>
Bloom Filter

Bloom filter is a *probabilistic* data structure used for membership queries

• Given a bloom filter, we can ask: “Have we seen this k-mer before?”
• No false negatives.
• May have false positives (in practice 5% false positive rate)

k-mers with frequency =1 are useless (either error or cannot be distinguished from error), and can safely be eliminated.
K-Mer Analysis: Step 3 (All-to-All)

Reads

- Parse to k-mers
- Find extensions of k-mers, hash them & find their owners
- All-to-all communication of k-mers and extensions
- Use a threshold & find the high quality extensions of k-mers

- Keep track of the number of occurrences of each extension for each k-mer

<table>
<thead>
<tr>
<th>P_0</th>
<th>Parse to k-mers</th>
<th>Hash k-mers</th>
<th>Received k-mers &amp; extensions</th>
<th>Local set</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td></td>
<td>Hash k-mers</td>
<td>Received k-mers &amp; extensions</td>
<td>Local set</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>Hash k-mers</td>
<td>Received k-mers &amp; extensions</td>
<td>Local set</td>
</tr>
<tr>
<td>P_n</td>
<td></td>
<td>Hash k-mers</td>
<td>Received k-mers &amp; extensions</td>
<td>Local set</td>
</tr>
</tbody>
</table>

Extensions:
- ACCCA CT
- CTTAG CF
- AACCT TG
- CGCAT XA
- AGGCA AT
- GGTAG FF
- AAAAT TG
- CCCAT XX
- TTCCA GT
- TTTGC CA
- AACTT GG
- CTTTT CA
Heavy Hitters

Long-tailed distribution for genomes with repetitive content:
- The maximum count for any k-mer in the wheat dataset is **451 million**
- Our original scheme (SC’14) was “owner counts”, after an all-to-all
- Counting an item w/ 451 million occurrences alone is **load imbalanced**

**Solution:** Quickly identify high-frequency k-mers using minimal communication during the “cardinality estimation” step and treat them specially by using local counters.
• The de Bruijn graph of k-mers is represented as a hash table.
• A k-mer is a node in a graph ⇔ a k-mer is an entry (key) in the hash table.
• An edge in the graph connects two nodes that overlap in k-1 bases.
• The edges in the hash table can be stored efficiently by storing the extensions of the k-mers as their corresponding values.
• The connected components represent contigs.
Parallel De Bruijn Graph Construction

Graph construction, traversal, and all later stages are written in UPC to take advantage of its global address space

**Input:** k-mers and their high quality extensions

**Read k-mers & extensions**

**Store k-mers & extensions**

**Distributed Hash table**

**Key:** ACC  **Val:** GA
**Key:** ATG  **Val:** CA
**Key:** AAT  **Val:** GF
**Key:** TCT  **Val:** GA
**Key:** CCG  **Val:** FA
**Key:** CTG  **Val:** AT
**Key:** TGC  **Val:** FA

**Shared**

**Private**

**buckets**

**entries**

**Global Address Space**

Parallel De Bruijn Graph Construction

19
Aggregating Stores Optimization

1 aggregate remote transfer
Full local buffer

Local buffer for $P_0$

Buffer local to $P_0$

Distributed Hash table

$P_0$ stores the k-mers & extensions in its local buckets in a lock-free & communication-free fashion
Parallel Graph Construction Challenge

• Challenge 1: Hash table for de Bruijn graph is huge (at least multiple terabytes)
  – Solution: Distribute the graph over multiple processors.

• Challenge 2: Parallel hash table construction introduces communication and synchronization costs
  – Solution: Split the construction in two phases and aggregate messages → 10x-20x performance improvement.
Parallel De Bruijn Graph Traversal

Algorithm: Pick a random k-mer and expand connected component by consecutive lookups in the distributed hash table.
Parallel De Bruijn Graph Traversal

• **Algorithm:** Pick a random k-mer and expand connected component by consecutive lookups in the distributed hash table.

• Fine-grained, irregular, remote accesses. Need fine-grained parallelization.
  - Worst case: the result is a single very long chain (high-diameter graph).
  - Global address space and one-sided communication simplifies logic.
  - If multiple processors are working on the same connected component, they cooperate via a **lightweight synchronization protocol**.
Lightweight synchronization protocol

A processor picks a k-mer as seed and initiates a subcontig

**inactive**

- Found unused k-mer in any direction of the subcontig:
  - 1) Successfully adding forward/backward extension to subcontig,
  - 2) Mark k-mer as USED

**active**

Attached in a neighboring subcontig

- Reached both endpoints of a UU contig:
  - 1) Set state to COMPLETE and store UU contig
  - 2) Pick another random seed from hash table

- Found used k-mers in both directions of the sub-contig AND both of the neighboring sub-contigs are active:
  - 1) Set own state to ABORTED,
  - 2) Pick another random seed from hash table

**aborted**

- Found used k-mers in both directions of the subcontig AND one of the neighboring subcontigs is ABORTED:
  - 1) Attach that subcontig to local subcontig,
  - 2) Set that state to active

**complete**

- Reached both endpoints of a UU contig. 
  - 1) Set state to COMPLETE and store UU contig
  - 2) Pick another random seed from hash table
**Scaffolding: Main step maps reads to contigs**

- **Input**: a set of reads and a set contigs
- **Output**: detailed alignments of the reads onto the contigs

Alignment with (large) input data; use of hash table to find possible matches

<table>
<thead>
<tr>
<th>Read ID</th>
<th>Start-pos</th>
<th>End-pos</th>
<th>Contig ID</th>
<th>Start-pos</th>
<th>End-pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read i</td>
<td>1</td>
<td>4</td>
<td>Contig 1</td>
<td>152</td>
<td>155</td>
</tr>
<tr>
<td>Read i</td>
<td>130</td>
<td>150</td>
<td>Contig 2</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>Read j</td>
<td>1</td>
<td>150</td>
<td>Contig 3</td>
<td>101</td>
<td>250</td>
</tr>
</tbody>
</table>
1. Given a set of reference sequences (contigs), build an index of them using seeds (substrings) of fixed length s.

2. Given a query sequence (read), extract substrings of length s and look those up in the reference index → locate candidate contigs to be aligned.

3. Perform an extension algorithm (e.g. Smith-Waterman) on the read and the candidate contig to obtain detailed alignments.
Gap Closing

- Gaps found from alignment and information about distance between paired reads
- Leads to load balancing problems (work stealing)
Extending HipMer for Metagenomes

- **Primary change in contig generation on k-mer graph**
  - Iterative contig generation, from small to large k-mer size
  - Small k: low coverage genomes
  - Large k: high-coverage genomes
  - Added/modified steps in HipMer: merging bubbles, iterative pruning, local assembly

- **Some changes to scaffolding**
  - Looping over scaffolding
  - Opportunities for future improvements

- **Scaffolding steps omitted from figure for simplicity**
Quality Metrics

- A goal of this milestone was to identify good quality metrics
- Want to simultaneously maximize contig length and minimize errors
  - Sort output contigs by length and find the halfway point.
  - Several standard metrics, simplest is N50.
  - \( N50 \) is the minimal contig length \( X \) such that contigs of length at least \( x \) account for at least 50\% of the total assembly length

```
120
90  Total assembly length = 500
90
50  First 3 contigs account for over half
50
50  N50 = 90 (length of 3rd contig)
50
```
Quality Metrics: Error-free contiguity

Contiguity

- **N50** is the contig length such that using longer or equal length contigs produces at least half (50%) the bases of the assembly.

- **NG50** is the contig length such that using longer or equal length contigs produces half (50%) the bases of the reference genome. This metric could be computed only if the reference is given.

Error-free contiguity

- **NGA50** similar to NG50, but uses lengths of aligned blocks rather than contigs. If a contig has a misassembly with respect to the reference, we break the contig into smaller pieces. Also, if a contig has bases that don't align to the reference, they are not counted in NGA50.

- **Median NGA50**: median NGA50 across all genomes for which NGA50 can be computed (requires sufficient coverage of genome)

We will use NGA50 and Median NGA50 for error-free contiguity
Results for low complexity data (mg64)

Low complexity dataset - only 64 genomes
All assemblers find most genomes, with low errors
MetaHipMer has more misassemblies due to small number of species (recall this is not normalized to assembly length)
MetaHipMer has fewer mismatches, greater predicted genes and better NGA50

<table>
<thead>
<tr>
<th>Genome Statistics</th>
<th>MetaHipMer</th>
<th>metaSPAdes</th>
<th>MEGAHIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genome Fraction (%)</td>
<td>95.5</td>
<td>94.2</td>
<td>95.3</td>
</tr>
<tr>
<td>Misassemblies (↓)</td>
<td>456</td>
<td>239</td>
<td>309</td>
</tr>
<tr>
<td>Mismatches per 100 kpb (↓)</td>
<td>45.6</td>
<td>99.3</td>
<td>71.9</td>
</tr>
<tr>
<td>Median NGA50</td>
<td>95129</td>
<td>89288</td>
<td>58468</td>
</tr>
<tr>
<td>Predicted Genes</td>
<td>202228</td>
<td>191307</td>
<td>200367</td>
</tr>
</tbody>
</table>
Error free contiguity (NGA50) on mg64
### Results for medium complexity data (CAMI)

<table>
<thead>
<tr>
<th>Genome Statistics</th>
<th>MetaHipMer</th>
<th>metaSPAdes</th>
<th>MEGAHIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genome Fraction (%)</td>
<td>66.3</td>
<td>68.7</td>
<td>73.0</td>
</tr>
<tr>
<td>Misassemblies (↓)</td>
<td>2031</td>
<td>1210</td>
<td>1579</td>
</tr>
<tr>
<td>Mismatches per 100 kpb (↓)</td>
<td>80.3</td>
<td>152.6</td>
<td>100.1</td>
</tr>
<tr>
<td>Median NGA50</td>
<td>24906</td>
<td>14219</td>
<td>16123</td>
</tr>
<tr>
<td>Predicted Genes</td>
<td>632459</td>
<td>635289</td>
<td>637085</td>
</tr>
</tbody>
</table>

- Medium complexity dataset - 225 genomes
- Lower coverage than mg64, with more errors
- MetaHipMer has a lower genome fraction, but better NGA50 than the others
Results for high complexity data (MC04)

<table>
<thead>
<tr>
<th>Genome Statistics</th>
<th>MetaHipMer</th>
<th>metaSPAdes</th>
<th>MEGAHIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genome Fraction (%)</td>
<td>28.6</td>
<td>32.7</td>
<td>32.3</td>
</tr>
<tr>
<td>Misassemblies (↓)</td>
<td>12730</td>
<td>9014</td>
<td>17479</td>
</tr>
<tr>
<td>Mismatches per 100 kpb (↓)</td>
<td>27.4</td>
<td>70.3</td>
<td>67.9</td>
</tr>
<tr>
<td>Median NGA50</td>
<td>15098</td>
<td>10540</td>
<td>12744</td>
</tr>
<tr>
<td>Predicted Genes</td>
<td>1386760</td>
<td>1381479</td>
<td>1355607</td>
</tr>
</tbody>
</table>

- High complexity dataset - 800 genomes
- Low coverage from all assemblers due to challenging nature of dataset
- High rates of misassembly
- MetaHipMer has a lower genome fraction, but better NGA50 than the others
MetaHipMer Performance (preliminary results)

- **Performance on a mock community**
  - metaSPAdes: **11.5 hours** on a 32 core machine with 500 GB of RAM
  - metaHipMer: **17 minutes** on 80 Edison nodes (~2K cores) – **41x faster**

- metaSPAdes can’t scale to the massive datasets (up to 80x larger)
  - For assembling TaraOceans dataset (8 TBytes, 80x larger), it would take **38 days**, on a machine with **40 TBytes** of memory.

- metaHipMer modules scale to full machine!
  - Assuming 40% efficiency, the TaraOceans dataset could be assembled in **< 1 hour** using full Edison
Summary on MetaGenomeAssembly

• **Assembly is an HPC problem**
  – (Meta)HipMer can handle previously unassembled data sets

• **Hardware support**
  – High injection rate RDMA communication; low latency; remote atomics
  – High bisection bandwidth (synchronous and asynchronous all-to-all)

• **Software support**
  – PGAS model for distributed data structures, one-sided communication
  – Hierarchical algorithms probably useful, but scaling well on/off node with UPC

• **Benchmarks and proxy applications**
  – NERSC-9 “Meraculous” benchmark is contig generation (graph/hash table)
  – Latency / bandwidth tests (“roofline like”) for remote atomics, async all-to-all

• **MetaHipMer is a parameterized toolkit for HPC genome analysis**
  – Quality results comparable to other state-of-art assemblers, but can solve problems they cannot due to memory/performance scaling
**ExaBiome: Exascale Solutions for Microbiome Analysis**

<table>
<thead>
<tr>
<th>Metagenome Assembly</th>
<th>Protein Clustering</th>
<th>Comparative Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph algorithms, Hash Tables, alignment (Smith-Waterman)</td>
<td>Machine learning (clustering), sparse linear algebra / graphs</td>
<td>Alignment, Machine learning (dimensionality reduction), linear algebra</td>
</tr>
<tr>
<td>Fine-grained comm., all-to-all, remote atomics and fast I/O</td>
<td>Fast barriers, subset reductions</td>
<td>All-to-all</td>
</tr>
</tbody>
</table>
Discovering Biosynthetic Gene Clusters

“A biosynthetic gene cluster is a physically clustered group of two or more genes in a particular genome that together encode a biosynthetic pathway for the production of a specialized metabolite (including its chemical variants)”

Input: Sparse matrix encoding genes and their nonzero pairwise similarities
Method: High-performance Markov clustering (HipMCL)
Desired scale: 10s of billions of genes, trillions of nonzero pairwise similarities
  • Use proteins rather than genes (constant factor)
  • Find connected components first (heuristic)
  • Requires supercomputers

MCL is the de-facto algorithm in community for finding gene/protein families
“...MCL is remarkably robust to graph alterations...”
DNA to Proteins

- DNA based on 4-character alphabet
- Three-letter codons represent one of the 20 regularly used amino acids
- Start and Stop codons mark the beginning/end of genes
Markov Cluster algorithm

Input: Adjacency matrix A (sparse)

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
Scalable Distributed Memory, Memory Efficient SpGEMM

- Parts of the result is produced and pruned
- Memory requirement can be significantly reduced by increasing $k$
- However, $A$ is needed to be broadcasted $k$ times
- With $k=20$: MCL ran on 64 nodes of Cori in about 20 minutes
Challenges in distributed-memory MCL

• “Small” test dataset from Joint Genome Institute
  – 47 Million proteins
  – 14 Billion interaction
  – 4.2 Trillion expected nonzeros in $A^2$
  – Memory requirement of naïve implementation: ~100 TB

• Ultimate dataset is 1000 times larger.

• Memory efficient SpGEMM algorithm being developed
  – Since the output is sparsified by column-wise pruning, we create $A^2$ part by part and prune on the fly to save memory.
  – Need to find the kth largest entry of each column at every iteration: Trivial in single node, harder in distributed memory.

• Interpretation of final clusters need distributed-memory connected components
  – Not a bottleneck: cheaper and only done once (not per iteration)
Machine Learning Mapping to Linear Algebra in General

- Logistic Regression, Support Vector Machines (SpMSpV)
- Dimensionality Reduction (e.g., NMF, CX/CUR, PCA)
- Clustering (e.g., MCL, Spectral Clustering)
- Graphical Model Structure Learning (e.g., CONCORD)
- Deep Learning (Convolutional Neural Nets)

Increasing arithmetic intensity

- Sparse Matrix-Sparse Vector Product (SpMSpV)
- Sparse Matrix-Dense Vector Product (SpMV)
- Sparse Matrix Times Multiple Dense Vectors (SpMM)
- Sparse-Sparse Matrix Product (SpGEMM)
- Dense Matrix Vector Product (BLAS2)
- Sparse-Dense Matrix Product (SpDM3)
- Dense Matrix Matrix Product (BLAS3)
ExaBiome: Exascale Solutions for Microbiome Analysis

**Metagenome Assembly**
- Graph algorithms, Hash Tables, alignment (Smith-Waterman)
- Fine-grained comm., all-to-all, remote atomics and fast I/O

**Protein Clustering**
- Machine learning (clustering), sparse linear algebra / graphs
- Fast barriers, subset reductions

**Comparative Analysis**
- Alignment, Machine learning (dimensionality reduction), linear algebra
- All-to-all
Metagenome composition description

Three approaches to describe and analyze the metagenome sequencing dataset:
1) taxonomic composition-based (who is in the metagenome)
2) functional annotation-based (what can this metagenome do)
3) k-mers (shingles) composition-based (i.e., shingle frequency vector-based)

K-mer based approach (MASH) is:
— More resilient to errors /degradation of DNA, coverage bias, composition, short and redundant fragments

Taxonomic (GOTTCHA) or functional approaches:
— Fast and memory efficient; also gives some semantic information about differences

“features”
MASH – an alternate MinHash based approach

• “...extends MinHash dimensionality-reduction technique to include a pairwise mutation distance and $P$ value significance test, enabling the efficient clustering and search of massive sequence collections...”

MASH distance is sufficiently accurate

MASH pros and cons

+ REALLY fast (just four simple consecutive steps).
  → K-mer extraction -> MinHash -> sketch -> distance

+ Reduces metagenome data size significantly
  → |sketch| << |metagenome|

+ Can cluster metagenomes with a lot of unknowns (un-annotatable)

  → There is a threshold in k-mer and sketch sizes which affects the tradeoff between specificity/sensitivity and the computational complexity/space requirements
  → The threshold can be estimated if the divergence between metagenomes can be roughly approximated

  → Provides no insight into the genome taxonomic or functional composition
GOTTCHA - Genome Signature Discovery

- GOTTCHA DB
- NCBI RefSeq (or a custom) genomic database
- Alignment process
- Taxonomic assignment #1
- Taxonomic assignment #2
- Taxonomic assignment #N
- Distance (between taxonomic abundance vectors)
- Distance Matrix
- Metagenome #1
- Metagenome #2
- ...
GOTTCHA – an interactive comparative visual display
GOTTCHA2 implementation

• Developed at LANL, will be released with GPL-2
• The database creation tool is not yet publicly available, coded in MPI C/C++ (we distribute binary database files)
• The taxonomic assignment pipeline is GPL-2 open sourced https://github.com/poeli/GOTTCHA2
• Currently relies on BWA for alignment https://github.com/lh3/bwa, distributed under GPL-3, we are working on the MerAligner integration from MetaHipMer
Summary on Post-Assembly Analytics

Finding gene clusters related groups (ML – clustering)
- Large graph / sparse matrix problems (sparse matrix-matrix multiply)
- Needs memory-efficient as well as scalable distributed memory approach

• Hardware support (preliminary)
  - High bisection bandwidth; low latency; fast I/O

• Software support
  - Sparse matrix and graph libraries (e.g., Graph BLAS approach)
  - MPI (bulk synchronous so far)

Comparative metagenome analysis (ML – dimensionality reduction)
- Dominated by alignment; see MerAligner discussion
- Also statistical (counting) and sparse matrix algorithms
# ExaBiome: Exascale Solutions for Microbiome Analysis (1.2.1.20)

## Application Domain

- **Application Area:** Microbiomes are integral to the environment, health, and biomanufacturing. They occur in communities that are collected and sequenced as a group, called metagenomes. These metagenomes lead to some of the most computationally demanding problems in bioinformatics, including assembly, protein clustering and comparative whole-metagenome analysis.

- **Challenge Problem:**
  - **4 Year:** Provide scalable tools for and complete assembly and analysis of metagenome data in SRA and IMG.
  - **10 Year:** Perform high quality assembly and analysis of 1 million metagenomes on an early exascale system to reveal microbial functions for environmental remediation and other applications.

## Physical Models and Code(s)

- **Physical Models:** Assembly of genomes using DNA sequence fragments from metagenomes. Clustering of genes culled from metagenomes to identify novel protein families. Comparative analysis across whole metagenomes.

- **Codes:** (Meta)HipMer, HipMCL, Mash, GOTCCHA, Combinatorial BLAS

- **Motifs:** Graph Traversal, Sparse Matrices, Dynamic Programming, (possibly Graphical Models)

## Partnerships

**Co-Design Centers:**
- ExaGraph (if funded) and possibly AMReX

**Software Technology Centers:**
- Programming: PAGODA, PROTEAS, Legion, ExaMPI and OMPI-X
- Performance/Systems: HPCToolkit, PAPI, Qthreads, ExaHDF5
- Libraries: xSDK4ECP, Sparse Solvers, Trilinos

## First Year Development Plans

**Develop high quality, scalable metagenome assembly**
- Extend HipMer with features for metagenomes, now called MetaHipMer (currently a research prototype)
- Evaluate and improve quality and scalability of MetaHipMer; compare to other metagenome assemblers
- Release MetaHipMer

**Use machine learning algorithms for metagenome analysis**
- Develop and demonstrate HipMCL clustering algorithm for genes from metagenomes
- Evaluate use of MerAligner (a phase in MetaHipMer) for GOTTCHA’s metagenome characterization

## Application Projects:
- Cancer, NWChem