Introduction to BLAST

PowerPoint by Ananth Kalyanaraman
School of Electrical Engineering and Computer Science
Washington State University
About the Presenter

- **Ananth** Kalyanaraman

  - Assistant Professor, School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA

  - Contact:
    - Email: ananth@eecs.wsu.edu
    - Website: [http://www.eecs.wsu.edu/~ananth](http://www.eecs.wsu.edu/~ananth)

- Ph.D., 2006, Iowa State University

- Research Interests:
  - Computational Biology and Bioinformatics
  - Parallel Algorithms and Applications
  - String Algorithms and Combinatorial Pattern Matching
Proliferation of Genomic Data

“An annotated collection of all publicly available nucleotide and amino acid sequences.”

GenBank:

• Doubles approximately 18 months

• > 190 billion bases

• Genomes:
  • Eukaryotes: ~200
  • Prokaryotes: ~600

• Metagenomic projects are a different league!
Topics for this Tutorial

- Review high-performance methods in computational genomics that belong to one of the following classes
  1. Compare one sequence vs. another sequence
     - Application: Sequence alignment
  2. Compare one sequence against many sequences
     - Application: Querying a database
Part I:
Sequence Alignment and Database Querying
Why Compare One Sequence to Another?

**Mutation** ➞ natural genetic variations

A genome mutating over generations

```
A C A G A G T A — A C
```

- Mutations are random events
- The effect of only some mutation events carry over to future generations
- Sequence comparison key for evolutionary studies

**Alignment** between $s_1$ and $s_2$

```
$\begin{align*}
\text{substitution} & : A C A G A G T A — A C \\
\text{deletion} & : A C A T A — T A G A C \\
\text{insertion} & :
\end{align*}$
```
How to Compare Two Sequences?

- **Problem:**
  - Given two sequences $s_1$ and $s_2$ over a fixed alphabet $\Sigma$, what is the set of variations that best describes the genetic transformation from $s_1$ to $s_2$ (or equivalently, from $s_2$ to $s_1$)?

- **Combinatorial Optimality**
  - Based on either maximizing an *alignment score* or minimizing *edit distance*
  - Standard dynamic programming techniques

- **Probabilistic Optimality**
  - Based on finding a most *probable* set of changes in aligning two sequences
  - Hidden-Markov Model (HMM) techniques
Two Important Types of Alignments

**Global**

Alignment between $s_1$ and $s_2$

- Needleman-Wunsch

Optimal global and local alignments can be computed in $O(|s_1| \cdot |s_2|)$ run-time and $O(|s_1| + |s_2|)$ space

**Preferred Applications**

For detecting two highly similar sequences (e.g., two homologous proteins)

**Local**

Alignment between a substring of $s_1$ and a substring of $s_2$

- Smith-Waterman

For detecting highly conserved regions (e.g., genes) between two sequences (e.g., genomes)
Need for a Fast Alignment Method

- What to do with a newly found gene candidate, $s_{new}$?
- Locate “similar” genes in GenBank

**One Approach: (database search)**

1. Concatenate all sequences in our genomic database into one sequence, say $s_d$
2. Compute the local alignment between $s_{new}$ and $s_d$
3. Report all “significant” local alignments

Run-time: $O(|s_d| \cdot |s_{new}|)$

Very long query time!!
Basic Local Alignment Search Tool (BLAST)

- Altschul et al. (1990) developed a program called BLAST to quickly query large sequence databases

- **Input:**
  - Query sequence q and a sequence database D

- **Output:**
  - List of all significant local alignment hits ranked in increasing order of *E*-value (aka *p*-value, which is the probability that a random sequence scores more than q against D).
0. **Preprocess:** Build a lookup table of size $|\Sigma|^w$ for all $w$-length words in $D$

$\Sigma=\{A,C,G,T\}$

$w = 2$

$\Rightarrow 4^2 (=16)$ entries in lookup table

Preprocessing is a one time activity
1. **Identify Seeds**: Find all \(w\)-length substrings in \(q\) that are also in \(D\) using the lookup table

2. **Extend seeds**: Extend each seed on either side until the aggregate alignment score falls below a threshold
   - **Ungapped**: Extend by only either matches or mismatches
   - **Gapped**: Extend by matches, mismatches or a limited number of insertion/deletion gaps

3. **Record** all local alignments that score more than a certain statistical threshold

4. **Rank and report** all local alignments in non-decreasing order of \(E\)-value
Illustration of BLAST Algorithm

Ungapped Extension

Gapped Extension
(over a band of diagonals)
## Different Types of BLAST Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Query</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>blastn</td>
<td>nucleotide</td>
<td>nucleotide</td>
</tr>
<tr>
<td>blastp</td>
<td>protein/peptide</td>
<td>protein/peptide</td>
</tr>
<tr>
<td>blastx</td>
<td>nucleotide</td>
<td>protein/peptide</td>
</tr>
<tr>
<td>tblastn</td>
<td>protein/peptide</td>
<td>nucleotide</td>
</tr>
<tr>
<td>tblastx</td>
<td>nucleotide</td>
<td>nucleotide</td>
</tr>
</tbody>
</table>

What if the Database Does Not Fit in the Main Memory?

Darling et al. (2003) show the effect by performing a blastn search when run on a system with 128 MB RAM. The increase in run-time is due to I/O.

Source: Darling et al. (2003)
HPC for BLAST

- Sequential BLAST is suitable for small number of queries
- HPC solutions for BLAST were developed to cater to large number of queries and also to address the rapid growth in database sizes
- We will review two HPC solutions for BLAST:

1. **mpiBLAST**: 

2. **ScalaBLAST**: 
mpiBLAST

- Input
  - Set of Queries, $Q=\{q_1, q_2, \ldots, q_m\}$, and
  - Database $D=\{s_1, s_2, \ldots, s_n\}$

- Let $p$ denote the number of processors, $M=\Sigma_{1 \leq i \leq m}|q_i|$, and $N=\Sigma_{1 \leq i \leq n}|s_i|$

- Algorithm follows the master-worker paradigm (1 master, $p-1$ workers)

- Assumption:
  - $Q$ is small enough to fit in the main memory of each worker

- Preferred:
  - Each worker processor has access to a local disk storage supporting contention-free local I/O
## mpiBLAST: The Parallel Algorithm

<table>
<thead>
<tr>
<th>Master</th>
<th>Worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>- The database D is <strong>fragmented</strong> into numerous disjoint pieces: ( F={f_1, f_2, \ldots, f_k}, \ k \gg p )</td>
<td>- Each worker ( p_i ) reads a subset ( F_i ) of ( F ) into its local storage, s.t., ( F=\bigcup_{1\leq i \leq p-1} F_i )</td>
</tr>
<tr>
<td>- The master processor broadcasts all queries in Q to workers</td>
<td>- Each worker sends the list of its local fragments to the master for housekeeping, and also reports that it is <em>idle</em></td>
</tr>
<tr>
<td>- The master processor records the list of “owners” for each database fragment</td>
<td></td>
</tr>
<tr>
<td>- The master then marks all fragments as <em>unassigned</em></td>
<td></td>
</tr>
</tbody>
</table>

Time
mpiBLAST: Algorithm …

**Master**

- The master assigns each database fragment to one worker. The fragment and order in which to assign is dynamically determined in a “greedy” fashion, as follows:
  - Each $p_i$ is allocated all its unique fragments first
  - Once such unique fragments are exhausted, a fragment $f$ is assigned to $p_i$, if $f \in F_i$ and $f$ is duplicated in least number of other workers
  - Finally, the remaining unassigned fragments are assigned to workers in decreasing order of their degrees of duplication

- The master processor ranks and outputs the hits for each BLAST query

**Worker**

- Each worker processor searches (ie., performs serial BLAST of $Q$ against) a database fragment assigned by the master.
  - If a fragment is not present in the local storage, it is copied from the corresponding worker that has it

- After searching each fragment, the results are communicated to the master processor
mpiBLAST: Run-time

“Green Destiny”:
- Beowulf cluster with a 100 Mb/s Ethernet
- Each compute node has a 667 MHz TM5600 CPU, 640 MB RAM, and a 20 GB local hard drive

- Database size is 5.1 GB
- Super-linear speedup observed as more memory becomes available for caching a bigger chunk of the local database fragments
- However, efficiency drops because of serial processing of output (during the final reporting step)

Source: from Darling et al. (2003)
mpiBLAST: Recent Improvements and Updates

- Parallel I/O for output processing (*mpiBLAST-PIO*)
  - (Local sorting + global merging) for all output records corresponding to each query
  - Very high scalability
    - Paper in this SC08 reports linear scaling on 32K BlueGene/L processors!

- [http://mpiblast.lanl.gov/](http://mpiblast.lanl.gov/)
ScalaBLAST: Main Ideas

- Removes I/O dependency by loading the entire target database into (distributed) memory

- All processors can access the entire database through *Global Array*, which is an interface for non-uniform memory access

- A query is evaluated entirely by a single processor group to avoid the serialization of reporting results later

- Supports layered parallelism:
  - The work related to each query is shared by processors in a MPI *process group* (compute nodes of an SMP node)
  - The query list itself is partitioned among the process groups
ScalaBLAST: Data and Processor Organization

An example with 8 processors:

Process Group

Global Array (distributed)

$$D$$

$$Q$$

$$g_0$$

$$g_1$$

$$g_2$$

$$g_3$$

$$m_0$$

$$m_1$$

$$m_2$$

$$m_3$$

$$p_0$$

$$p_1$$

$$p_2$$

$$p_3$$

$$p_4$$

$$p_5$$

$$p_6$$

$$p_7$$

$$m_0$$ memory

$$m_1$$ memory

$$m_2$$ memory

$$m_3$$ memory
ScalaBLAST: The Algorithm

1. Both the database $D$ and query list $Q$ are evenly *partitioned* across processor groups over their sizes.

2. In each process group $g_i$, the corresponding $p_{0}'$ and $p_{1}'$ perform BLAST search on the local query list, one query at a time. For a given query $q$,
   - $p_{0}'$ performs the BLAST operation on the first half on the database while $p_{1}'$ performs BLAST operation on the second half
   - Results for $q$ are then trivially merged, ranked and reported by one of the processors

3. Each process element posts a non-blocking request for the next portion of database resident in a remote memory, *before* starting to compute BLAST operation on the current portion of database. This *pre-fetching masks communication overhead* with computation.
ScalaBLAST: Performance Results

- **Database**: 1.5 million protein sequences ≈ 503 characters
- **Query**: 1,000 sequences of total size 709 Kbytes
- **Experimental Platforms**:
  - **MPP2**, a distributed memory machine with 1.5 GHz Itanium II processors and Quadrics Elan-4 interconnect, 6 to 8 GB RAM/per node
  - **SGI Altix**, an SMP with 128 1.5 GHz Itanium II processors and with 256 GB.

### Phase-wise Run-time

<table>
<thead>
<tr>
<th></th>
<th>Setup %</th>
<th>Query %</th>
<th>Output %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Source**: Oehman and Nieplocha (2006)
More information about ScalaBLAST

Selected Bibliography for Alignment Topics

Papers

Books
Selected Bibliography for BLAST Related Topics

Serial BLAST

HPC BLAST
NCBI BLAST - Web Resources

- NCBI BLAST Webpage:

- For a comprehensive list of BLAST related references: