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

- Ph.D., 2006, Iowa State University

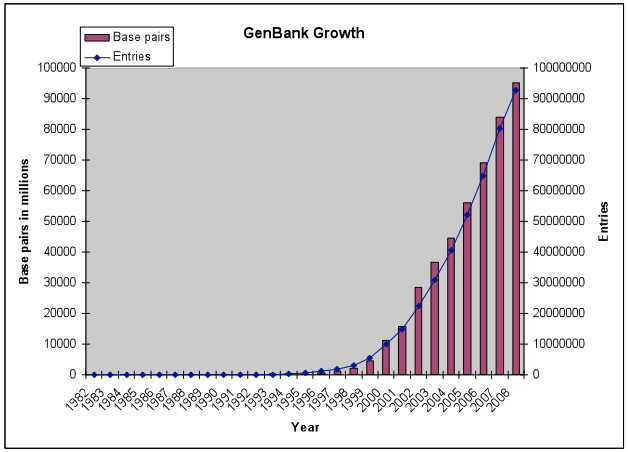
- Research Interests:
  - Computational Biology and Bioinformatics
  - Parallel Algorithms and Applications
  - String Algorithms and Combinatorial Pattern Matching

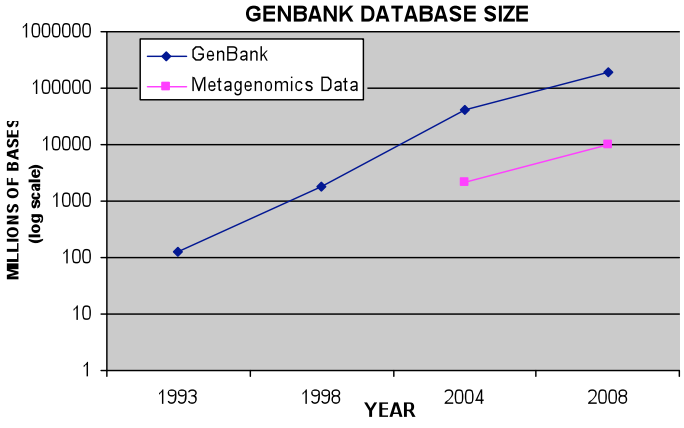
Proliferation of Genomic Data

"*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

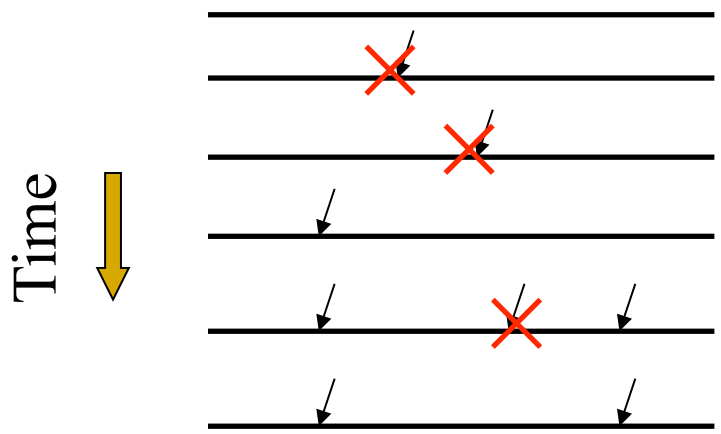
- Review high-performance methods in computational genomics that belong one of the following classes
  1. Compare one sequence vs. another sequence
     - Application: Sequence alignment
  2. Compare one sequence against many sequences
     - Application: Querying a database

Part I: Sequence Alignment and Database Querying

Why Compare One Sequence to Another?

- *Mutation* ➔ natural genetic variations

A genome mutating over generations



- 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

s_1: A C A G A G T A – A C

s_2: A C A T A – T A G A C

substitution deletion insertion

How to Compare Two Sequences?

- **Problem**:
  - Given two sequences s_1 and s_2 over a fixed alphabet Σ, 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)?

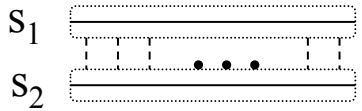
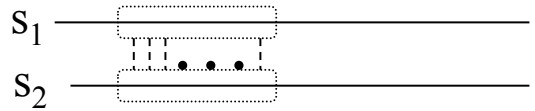
Combinatorial Optimality

- 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

		Preferred Applications
Global Needleman-Wunsch	Alignment between s_1 and s_2 	For detecting two highly similar sequences (eg., two homologous proteins)
Local Smith-Waterman	Alignment between a substring of s_1 and a substring of s_2 	For detecting highly conserved regions (eg., genes) between two sequences (eg., genomes)

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

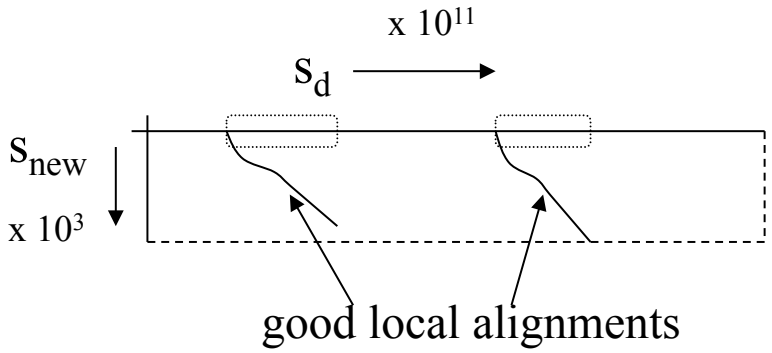
Need for a Fast Alignment Method

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

One-to-many

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: $O(|s_d|.|s_{new}|)$

⇩

Very long
query time !!

Basic Local Alignment Search Tool (BLAST)

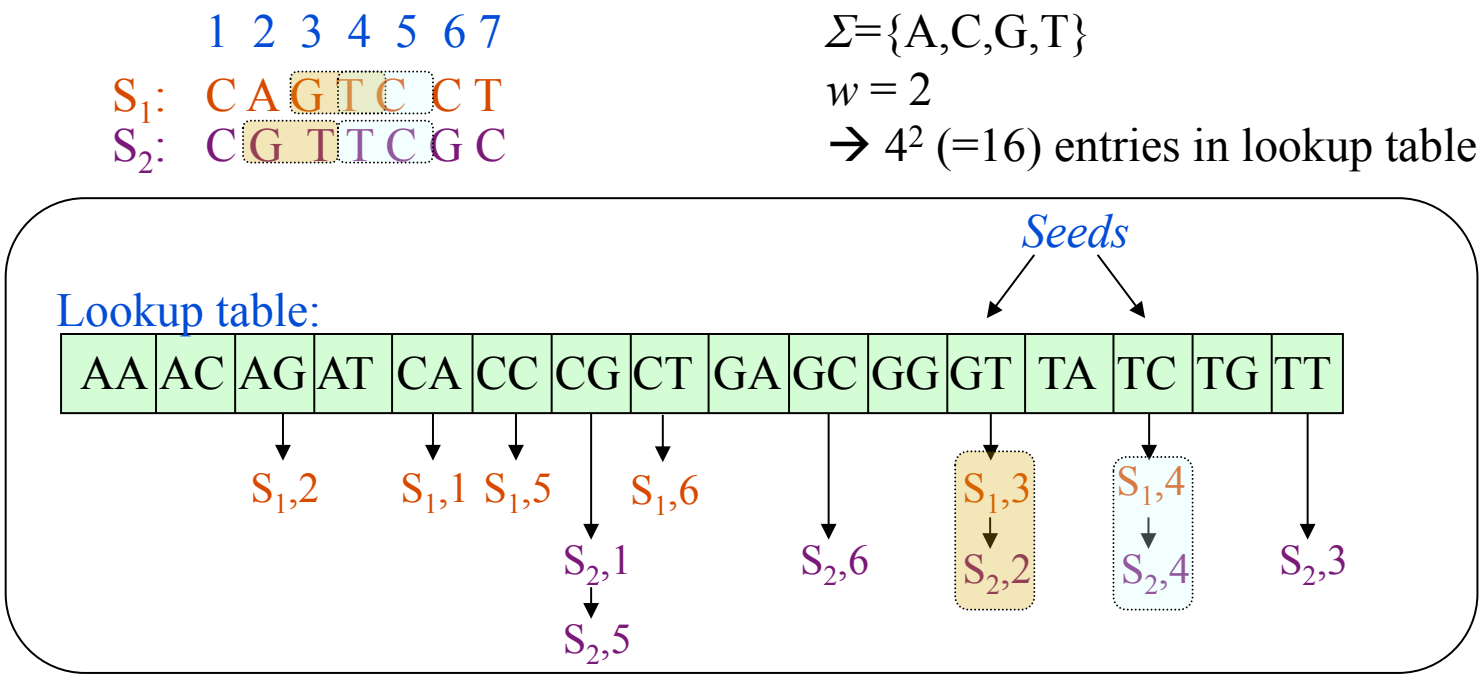
- Altschul *et al.* (1990) developed a program called BLAST to quickly query large sequence databases

- **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).

BLAST Algorithm

0. Preprocess: Build a *lookup table* of size $|\Sigma|^w$ for all w-length words in D



Preprocessing is a one time activity

BLAST Algorithm …

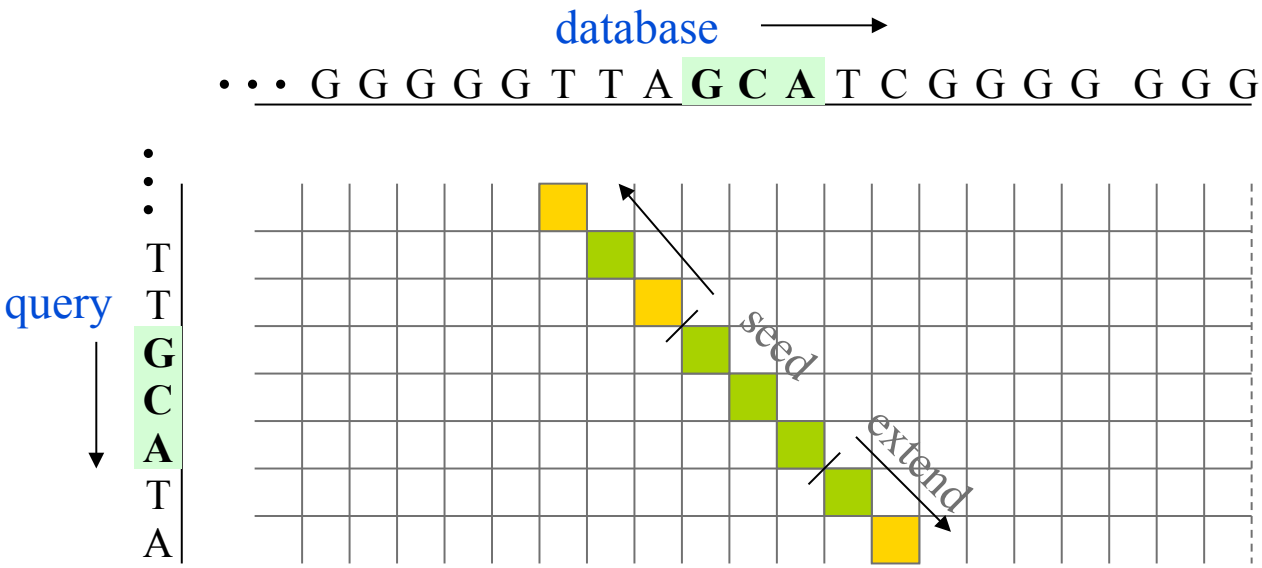
1. Identify Seeds: Find all *w*-length substrings in *q* that are also in D using the lookup table

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

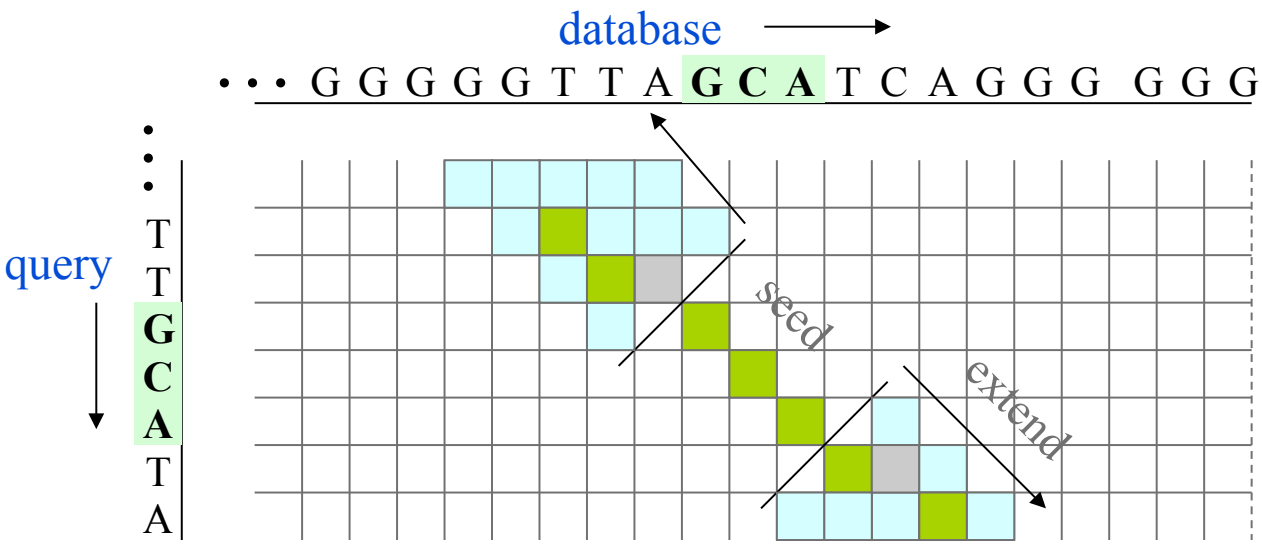
3. Record all local alignments that score more than a certain statistical threshold

4. Rank and report all local alignments in non-decreasing order of *E-value*

Illustration of BLAST Algorithm



Ungapped Extension



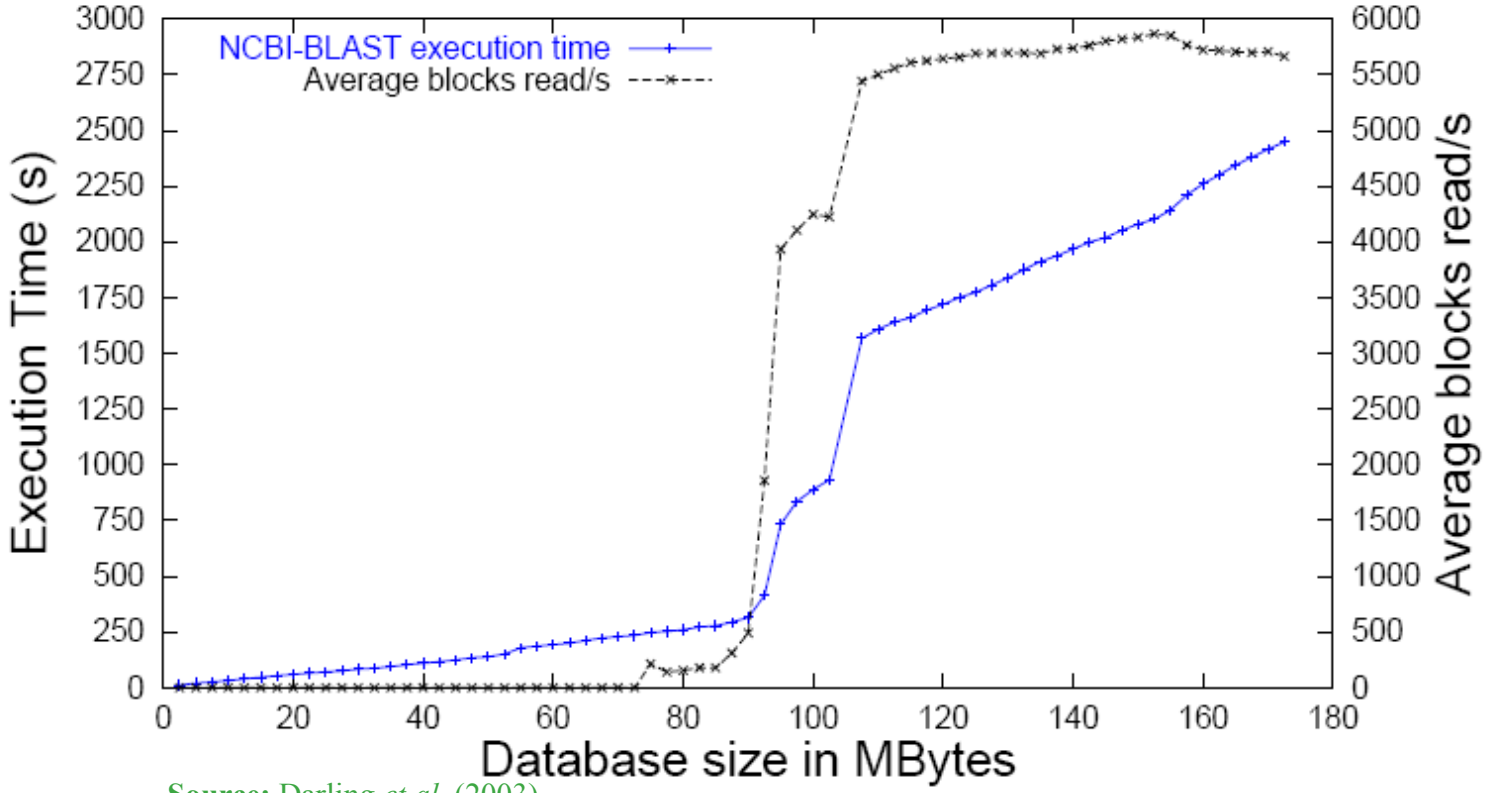
Gapped Extension (over a band of diagonals)

Different Types of BLAST Programs

Program	Query	Database
blastn	nucleotide	nucleotide
blastp	protein/peptide	protein/peptide
blastx	nucleotide	protein/peptide
tblastn	protein/peptide	nucleotide
tblastx	nucleotide	nucleotide

http://www.ncbi.nlm.nih.gov/blast

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



Source: Darling *et al.* (2003)

- 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 .

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**:

   Darling *et al.* (2003), "The Design, Implementation, and Evaluation of mpiBLAST", *Proc. ClusterWorld*.

2. **ScalaBLAST**:

   Oehmen and Nieplocha (2006), "ScalaBLAST: A Scalable Implementation of BLAST for High-Performance Data-Intensive Bioinformatics Analysis", *IEEE Transactions on Parallel and Distributed Systems*, 17(8):740-749.

mpiBLAST

- Input
  - Set of Queries, $Q=\{q_1,q_2,...,q_m\}$, and
  - Database $D=\{s_1,s_2,\ldots,s_n\}$
- Let *p* denote the number of processors, $M=\Sigma_{1\le i\le m}|q_i|$, and $N=\Sigma_{1\le i\le n}|s_i|$
- 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

Master

- The database D is *fragmented* into numerous disjoint pieces:
  $F=\{f_1,f_2,\ldots,f_k\},\ k>>p$
- The master processor broadcasts all queries in Q to workers
- The master processor records the list of "owners" for each database fragment
- The master then marks all fragments as *unassigned*

Time

Worker

- Each worker p_i *reads* a subset F_i of F into its local storage, s.t., $F=\mathrm{U}_{1\le i\le p-1}F_i$
- Each worker sends the list of its local fragments to the master for housekeeping, and also reports that it is *idle*

mpiBLAST: Algorithm …

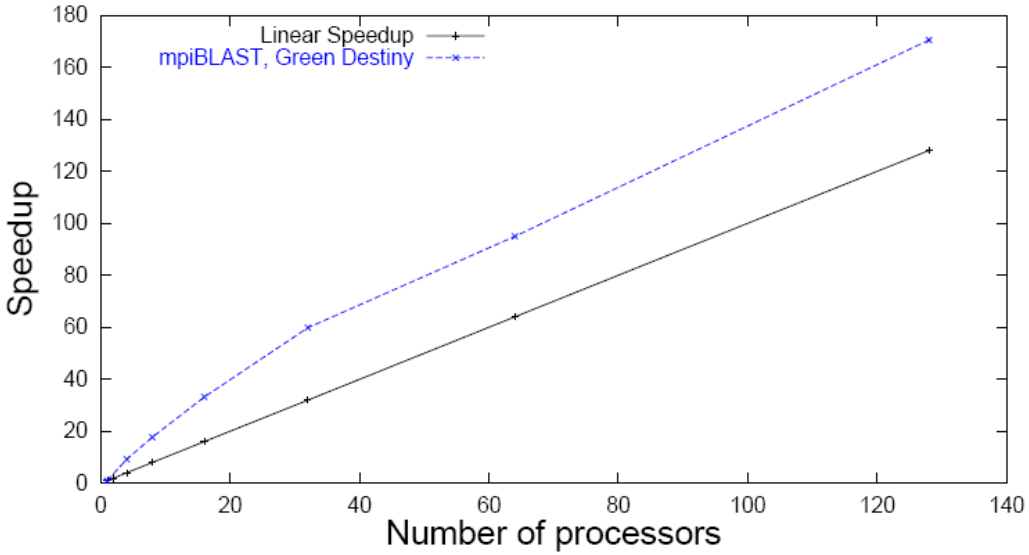
Master

- 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



Source: from Darling *et al.* (2003)

"Green Destiny":

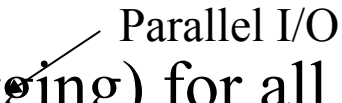
-Beowulf cluster with a 100 Mb/s Ethernet

-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)

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!



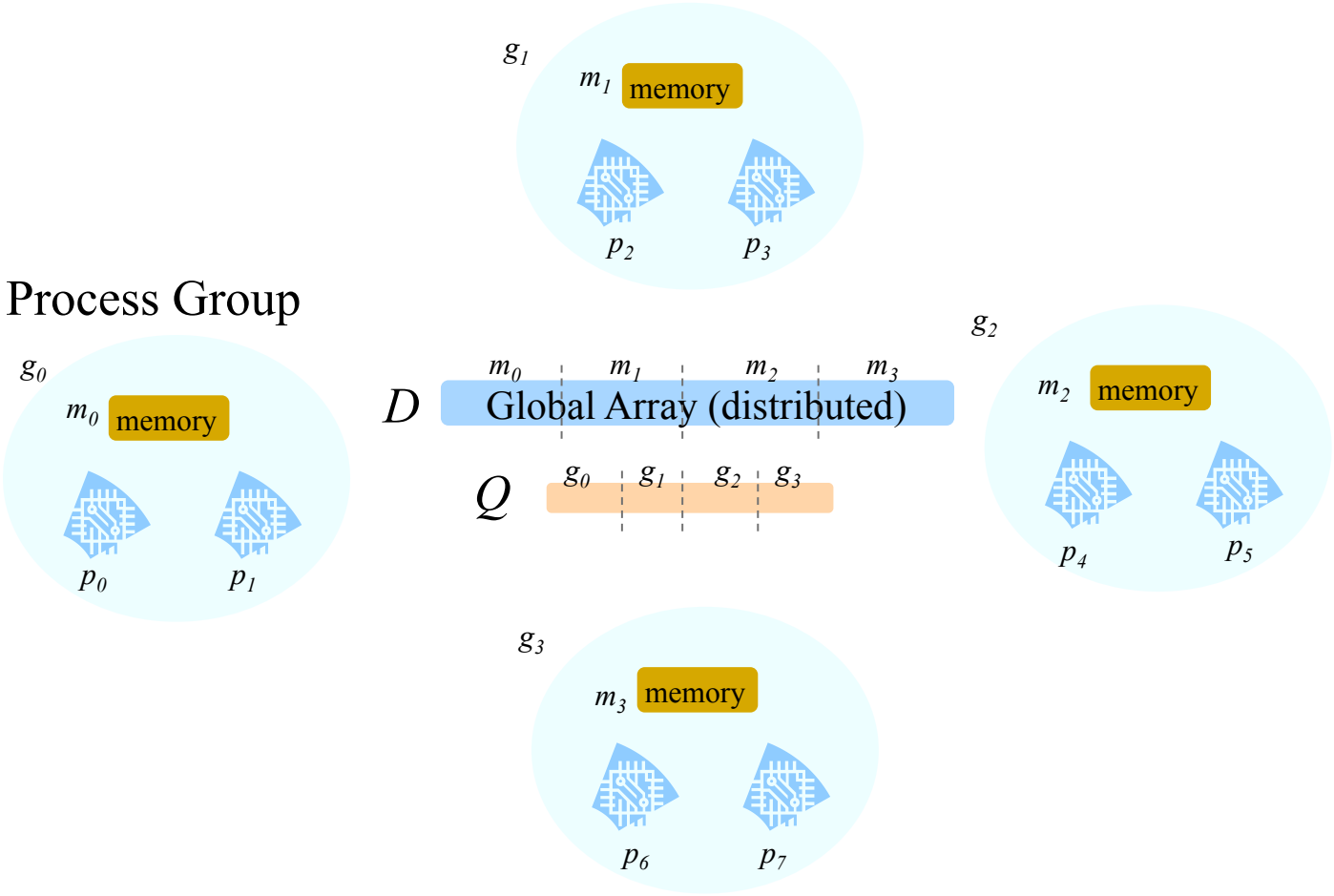
- *http://mpiblast.lanl.gov/*

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:



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

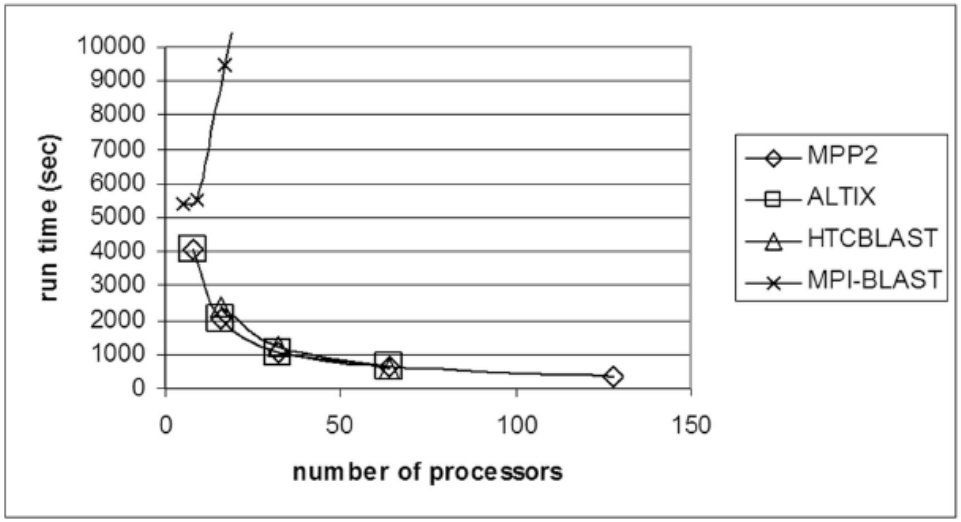
3. Each process element posts a non-blocking request for the next portion of database resident in a remote memory, *before* starting to compute BLAST operation on the current portion of database. This pre-fetching masks communication overhead with computation

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

	Setup %	Query %	Output %
|Q|=100 p=8	~ 2.5	~ 95	~ 2.5
|Q|=1000 p=8	< 0.1	~ 98.5	~ 1.4
|Q|=1000 p=32	< 0.3	~ 98.3	~ 1.5



Source: Oehman and Nieplocha (2006)

More information about ScalaBLAST

- *http://hpc.pnl.gov/projects/scalablast/*

Selected Bibliography for Alignment Topics

Papers

- S. Needleman and C. Wunsch (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Molecular Biology*, 48:443-453.
- D. Hirschberg (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341-343.
- T. Smith and M. Waterman (1981). Overlapping genes and information theory, *J. Theoretical Biology*, 91:379-380.
- O. Gotoh (1982). An improved algorithm for matching biological sequences. *J. Molecular Biology*, 162(3): 705-708.
- J. Fickett (1984). Fast optimal alignment. Nucleic Acids Research, 12(1):175-179.
- M.S. Gelfand *et al.* (1996). Gene recognition via spliced alignment. *Proc. National Academy of Sciences*, 93(17):9061-9066.
- A. Delcher *et al.* (1999). Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369-2376.
- X. Huang and K. Chao (2003). A generalized global alignment algorithm. *Bioinformatics*, 19(2):228-233.
- S. Rajko and S. Aluru (2004). Space and time optimal parallel sequence alignments. *IEEE Transactions on Parallel and Distributed Systems*, 15(12):1070-1081.

Books

- D. Gusfield (1997). Algorithms on strings, trees and sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge, London.
- J. Setubal and J. Meidanis (1997). Introduction to computational molecular biology. PWS Publishing Company, Boston, MA.
- B. Jackson and S. Aluru (2005). Chapter: “Pairwise sequence alignment” in Handbook of computational molecular biology, Ed. S. Aluru, Chapman & Hall/CRC Press.

Selected Bibliography for BLAST Related Topics

Serial BLAST

- S. Altschul *et al.* (1990). Basic Local Alignment Search Tool, *J. Molecular Biology*, 215:403-410.
- W. Gish and D.J. States (1993). Identification of protein coding regions by database similarity search. *Nature Genetics*. 3:266-272.
- T.L. Madden *et al.* (1996). Applications of network BLAST server. *Meth. Enzymol*. 266:131-141.
- S. Altschul, *et al.* (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389-3402.
- Z. Zhang *et al.* (2000). A greedy algorithm for aligning DNA sequences. *J. Computational Biology*, 7(1-2): 203-214.

HPC BLAST

- T. Rognes (2001). ParAlign: A parallel sequence alignment algorithm for rapid and sensitive database searches, Nucleic Acids Research, 29:1647-1652.
- R. Bjornson *et al. (*2002). TurboBLAST®: A parallel implementation of BLAST built on the TurboHub, *Proc. International Parallel and Distributed Processing Symposium.*
- A. Darling, L. Carey and W.C. Feng (2003). The design, implementation, and evaluation of mpiBLAST, *Proc. ClusterWorld.*
- D. Mathog (2003). Parallel BLAST on split databases, *Bioinformatics*, 19:1865-1866.
- J. Wang and Q. Mu (2003). Soap-HT-BLAST: High-throughput BLAST based on web services, *Bioinformatics*, 19:1863-1864.
- H. Lin *et al.* (2005). Efficient data access for parallel BLAST, *Proc. International Parallel and Distributed Processing Symposium*.
- K. Muriki, K. Underwood and R. Sass (2005). RC-BLAST: Towards a portable, cost-effective open source hardware implementation, *Proc. HiCOMB 2005.*
- M. Salisbury (2005). Parallel BLAST: Chopping the database, *Genome Technology*, pp 21-22.

NCBI BLAST - Web Resources

- NCBI BLAST Webpage:
  http://www.ncbi.nlm.nih.gov/BLAST/

- For a comprehensive list of BLAST related references:
  http://www.ncbi.nlm.nih.gov/blast/blast_references.shtml