



Efficient Data Handling in Large-Scale Sequence Database Searches

H. Lin[†], X. Ma[†], W. Feng^{*},
A. Geist[‡], and N. Samatova[‡]

[†]NCSU

^{*}Virginia Tech

[‡]ORNL

1



Outline

- Sequence database search
- Parallel BLAST background
- mpiBLAST & pioBLAST
- New release: mpiBLAST-pio
- GreenGene: search NT against NT practice (SC|05, StorCloud Demo)

2

Sequence Database Search is Critical for Biomedical Science

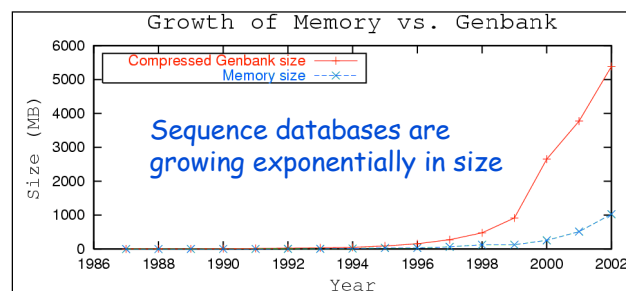
- Routinely used in biomedical research
 - Search similarities between query sequences and sequence database
 - Predict structures and functions of new sequences
- Analogous to web search engines (e.g. Google)

	Web Search Engine	Sequence DB Search
Input	Keyword(s)	Query sequence(s)
Search space	Internet	Known sequence database
Output	Related web pages	DB sequences similar to the query
Sorted by	Closeness & rank	Score (Similarity)

3

Challenge for Sequence DB Search

Sequence DB Search is Hampered by the Growing Gap between Sequence Growth and Processor Memory



Because of this gap: there is a lot of repeated I/O introduced by loading sequence data back and forth from the file system to the memory. This adversely affects the performance.

4

BLAST: At the Core of Sequence DB Search

- Widely used search tool:
 - Approximately 75%-90% of all compute cycles in life sciences are devoted to BLAST searches
- But, it is:
 - Computationally demanding, $O(n^2)$
 - Requires huge database to be stored in memory
 - Generates gigabytes of output file for large database searches
- Parallel BLAST as a means to address computational challenge

5

BLAST Parallelization: Query Segmentation

Queries

>Perilla Frutescens CDS 0001

```
TTGGTATCCACGGAAAGAGAGAGAAAATGTTGGGAATTTTCAGCGGAC
GTATAGTATCATTGCCGGAAGAGCTGGTGGCTGCCGGGAACC
```

>Perilla Frutescens CDS 0002

```
GGAGGGTGGCTGGTGGGTATTGGCGGCCCGACCGATCGCCCGGAC
CGACGGCTCCTGCCACCCGAACATGTGATAGAAAGGAQQQQQQQ
```

>Perilla Frutescens CDS 0003

```
TTTTTTTCTTGATGCTGAAATCTATCCAAACATCACCAGTCTCACGAG
TCCTTGACCAAATTCCTGCTTTCTGGCACAATCTGAAGCCCAAAGGC
```

Database

>gi|3123744|dbj|AB013447.1|AB013447

```
TTGGTATCCACGGAAAGAGAGAGAAAATGTTGGGAATTTTCAGCGGAC
GTATAGTATCATTGCCGGAAGAGCTGGTGGCTGCCGGGAACC
```

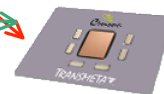
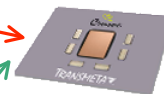
>gi|221778|dbj|D00026.1|HS2HSV2P4

```
GGAGGGTGGCTGGTGGGTATTGGCGGCCCGACCGATCGCCCGGAC
CGACGGCTCCTGCCACCCGAACATG
```

>gi|7328961|dbj|AB032155.1|AB032154S2

```
TTTTTTTCTTGATGCTGAAATCTATCCAAACATCACCAGTCTCACGAG
TCCTTGACCAAATTCCTGCTTTCTGGCACAATCTGAAGCCCAAAGGC
```

Worker Nodes



W. Feng et al. "mpiBLAST on the GreenGene Distributed Supercomputer", SC05

6

Pros and Cons of Query Segmentation

■ Advantages

- Low parallelization overhead
- Linear speedup when database fits into single processor memory

■ Disadvantages

- Suffers repeated I/O when database cannot fit into main memory
- Resource under-utilization / load imbalance when #queries smaller than or comparable to #processors

7

BLAST Parallelization: Database Segmentation

Queries

```
>Perilla Frutescens CDS 0001
TTGGTATCCACGGAAGAGAGAGAAAATGTTGGGAATTTTCAGCGGAC
GTATAGTATCATTGCCGGAAGAGCTGGTGGCTGCCGGGAACC

>Perilla Frutescens CDS 0002
GGAGGGTGGCTGGTGGGATTGGCGGCCCGACCGATCGCCCCGAC
CGACGGCTCCTGCCACCCGAACATGTGATAGAAAGGAQQQQQQQ

>Perilla Frutescens CDS 0003
TTTTTTTCTTGATGCTGAAATCTATCCAACATCACCAGTCCTCACGAG
TCCTTGACCAAATCTTGCTTTCTGGCACAATCTGAAGCCCAAAGGC
```

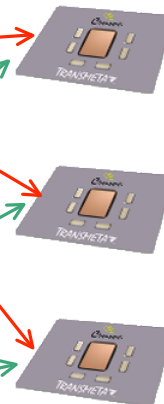
Database

```
>gi|3123744|dbj|AB013447.1|AB013447
TTGGTATCCACGGAAGAGAGAGAAAATGTTGGGAATTTTCAGCGGAC
GTATAGTATCATTGCCGGAAGAGCTGGTGGCTGCCGGGAACC

>gi|221778|dbj|D00026.1|HS2HSV2P4
GGAGGGTGGCTGGTGGGATTGGCGGCCCGACCGATCGCCCCGAC
CGACGGCTCCTGCCACCCGAACATG

>gi|7328961|dbj|AB032155.1|AB032154S2
TTTTTTTCTTGATGCTGAAATCTATCCAACATCACCAGTCCTCACGAG
TCCTTGACCAAATCTTGCTTTCTGGCACAATCTGAAGCCCAAAGGC
```

Worker Nodes



W. Feng et al. "mpiBLAST on the GreenGene Distributed Supercomputer", SC'05

8



Pros and Cons of Database Segmentation

- Advantages
 - Fitting large database into aggregate memory
 - Able to utilize large machines regardless of #queries
- Disadvantages
 - Higher parallel search overhead, local results need to be merged globally
- Challenge
 - Reduce result merging & processing overhead

9



mpiBLAST: A Specific Implementation of Database Segmentation

- Open-source parallel BLAST developed at LANL:
 - <http://mpiblast.lanl.gov> or <http://www.mpiblast.org>
- Increasingly popular: more than 10,000 downloads in less than 2 years
- Integrated with NCBI BLAST
- Based on database segmentation
- Performance
 - Achieves super linear speedup when using small # processors
 - **Problem:** overhead in data handling limits scalability

10

mpiBLAST System Design

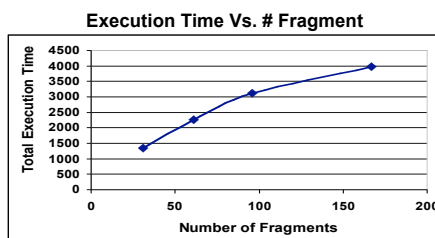
- Master-slave model: one master, p-1 workers
- Searching done in workers
 - Search all queries against a subset of DB frags
 - Generate partial results - meta data of alignments (ASN.1 format, include seq id, scores, etc.)
- Output processing done in master
 - Merge partial results from all workers
 - Fetch correspondent result sequence data
 - Compute and output alignments

11

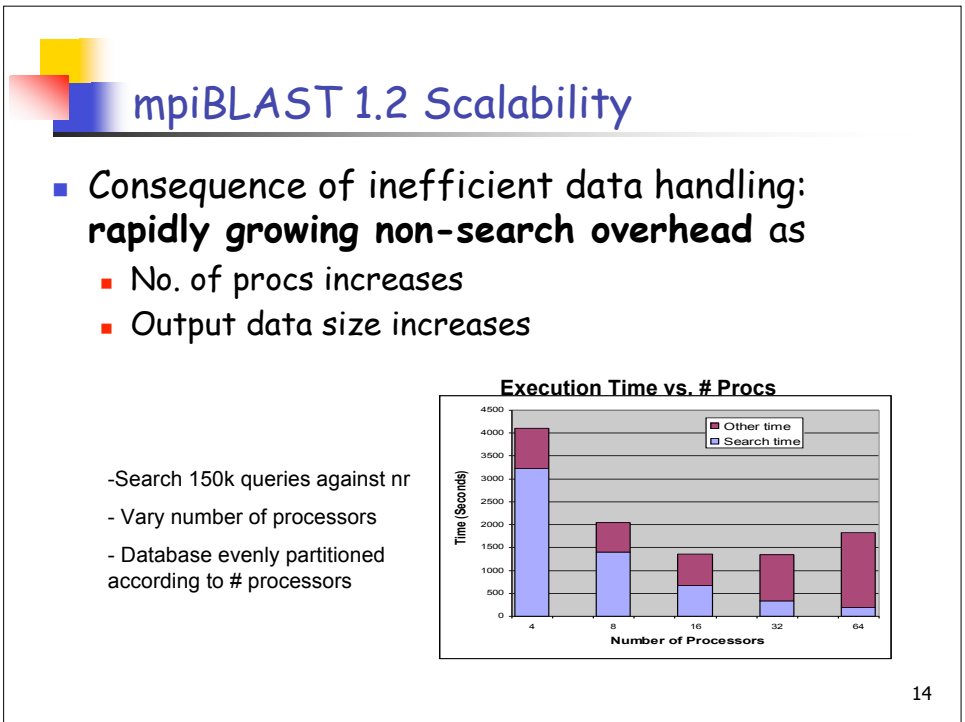
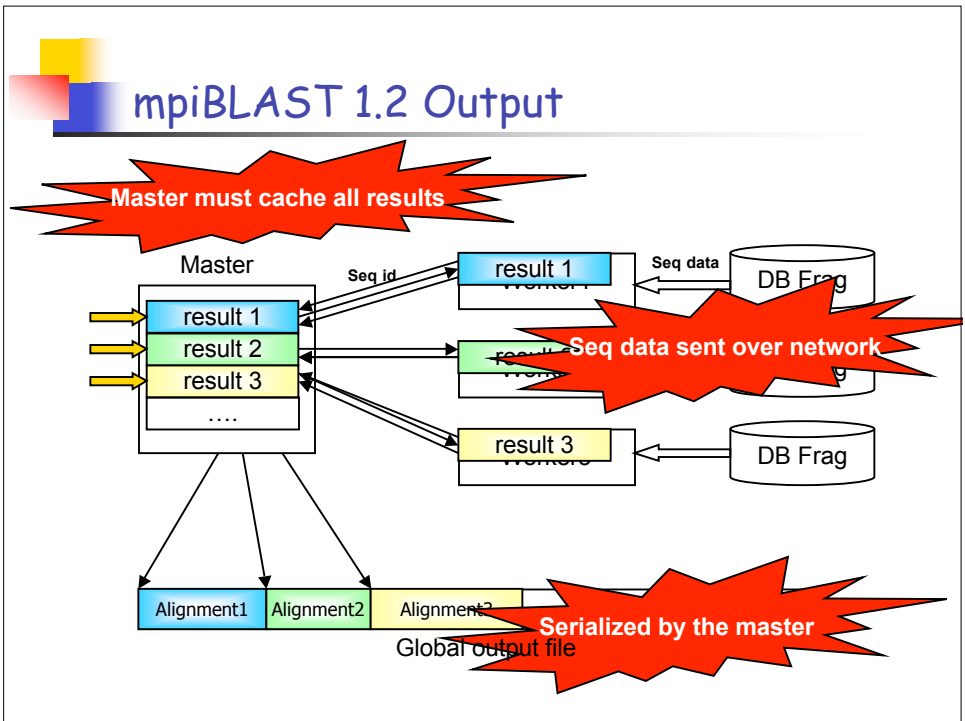
mpiBLAST 1.2 Input

- Databases partitioned **statically** before search
 - **Inflexible**
 - execution time sensitive to # fragments
 - re-partitioning required to use different # procs
 - **Management overhead**
 - generating large number of small files, hard to manage, migrate and share

Fragments sensitivity test
- Search 150k queries against nr database
- Using 32 processors



12



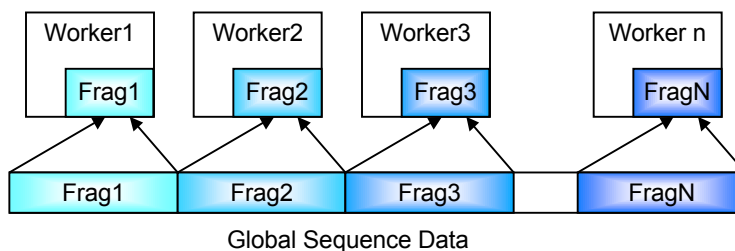
pioBLAST

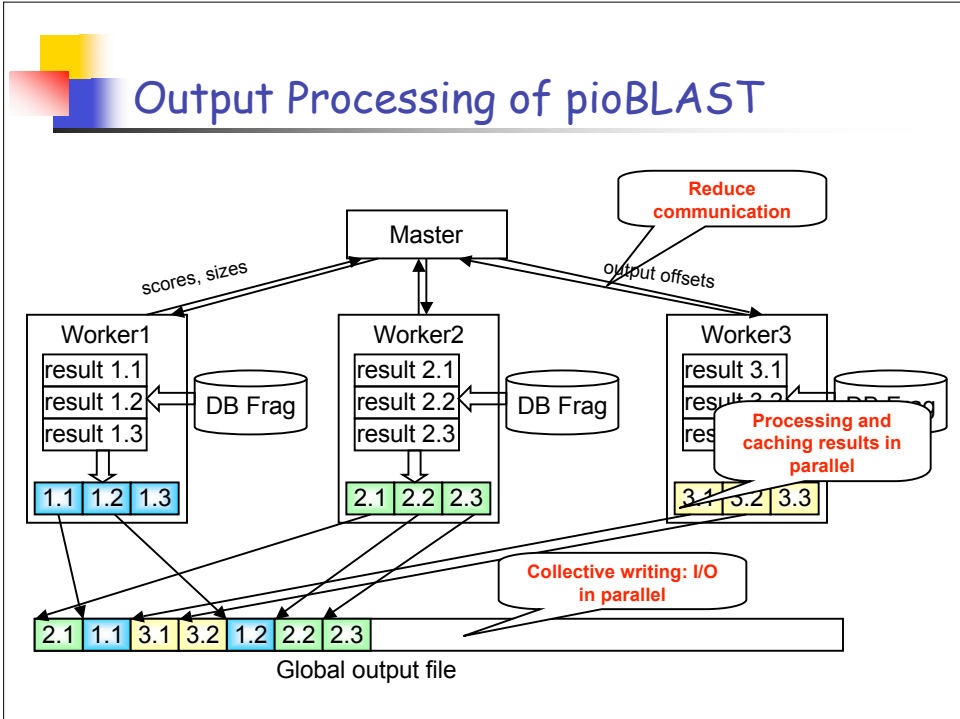
- Research prototype of efficient parallel BLAST developed at ORNL & NCSU
- Built on top of mpiBLAST1.2
- Apply **parallel/collective I/O** techniques
 - Enable dynamic partitioning
 - Parallel database input and result output
- Highly efficient **result processing**
 - Workers compute alignments in parallel
 - Workers buffer and write local output in parallel
 - Enhanced worker-master communication for reducing data transfer volume

15

Dynamic Partitioning of pioBLAST

- No pre-partitioning
 - One single database image to search against
- Virtual fragments generated dynamically at run time
 - Workers read inputs in parallel with MPI-IO interface
- Fragment size configurable at run time
 - Easily supports dynamic load balancing





mpiBLAST 1.2 vs. pioBLAST: Node Scalability

- Platform: SGI Altix at ORNL
 - 256 processors (1.5GHz Itanium2), 8GB memory/proc, XFS
- Database: nr (1GB)
- Node scalability
 - mpiBLAST: **non-search** overhead increases fast
 - pioBLAST: **non-search** time remains low

Search 150k NR queries on different #procs

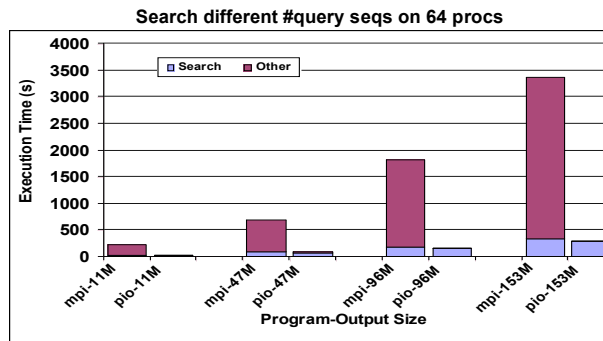
Program	No. of processes	Search time (s)	Other time (s)	Total time (s)
mpi-4	4	~3000	~1000	~4000
pio-4	4	~2800	~100	~2900
mpi-8	8	~1200	~800	~2000
pio-8	8	~1100	~100	~1200
mpi-16	16	~600	~800	~1400
pio-16	16	~500	~100	~600
mpi-32	32	~300	~1000	~1300
pio-32	32	~200	~100	~300
mpi-64	64	~100	~1500	~1600
pio-64	64	~100	~100	~200

Program-No. of processes

18

mpiBLAST 1.2 vs. pioBLAST: Output Scalability

- Same platform and database
- Varied query size to generate different output size



19

mpiBLAST Evolves: v1.4

- Exact e-value statistics
- Improved result processing
 - Reduce worker-master communication by packing partial biosequences along with ASN.1 results
 - Alleviate master bottleneck with query pipe-lining
- Not ready for the large DB search
 - Output processing still **serialized**
 - Partial results and result sequences data for **a single query** could be huge (gigabytes)
- Performance
 - Efficient in handling queries with **small** output
 - Hang or perform slow for queries with **large** output


20



mpiBLAST + pioBLAST = **mpiBLAST-pio**

- Highly efficient, open source parallel BLAST (available at <http://mpiblast.lanl.gov/>)
- Joint effort between mpiBLAST and pioBLAST research teams
- Current release based on mpiBLAST 1.4
 - Exact e-value statistics
 - Keep scheduling (query pipelining) and data distribution
- Efficient parallel output processing from pioBLAST
 - Worker compute and buffer local output in parallel
 - Non-collective parallel write to better support query pipelining
 - Modifications on NCBI BLAST less than 30 lines
 - Support all but anchor output formats

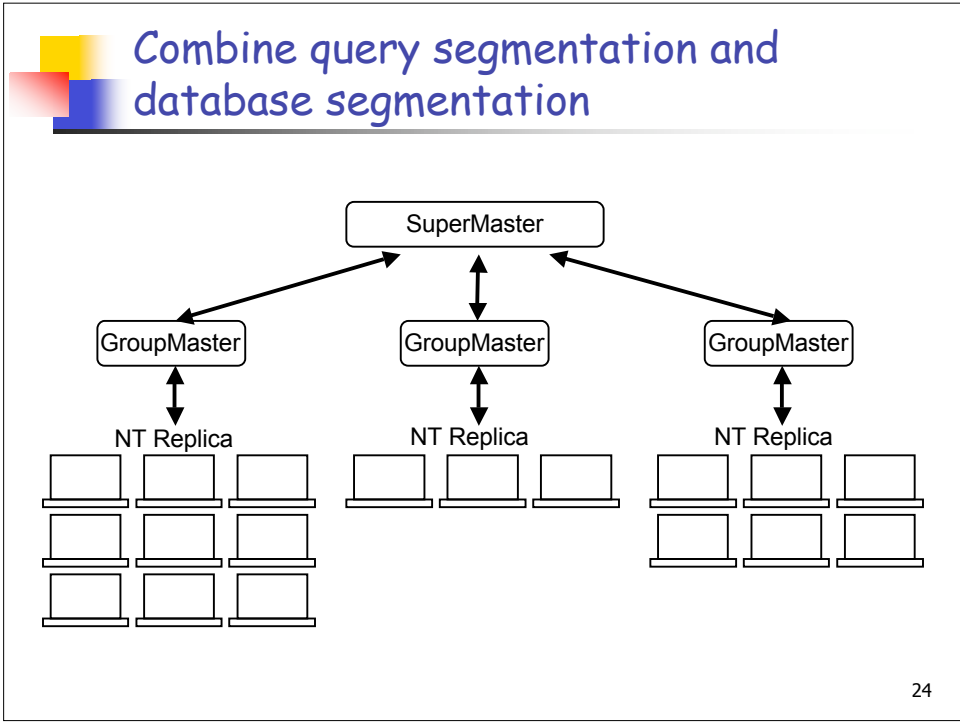
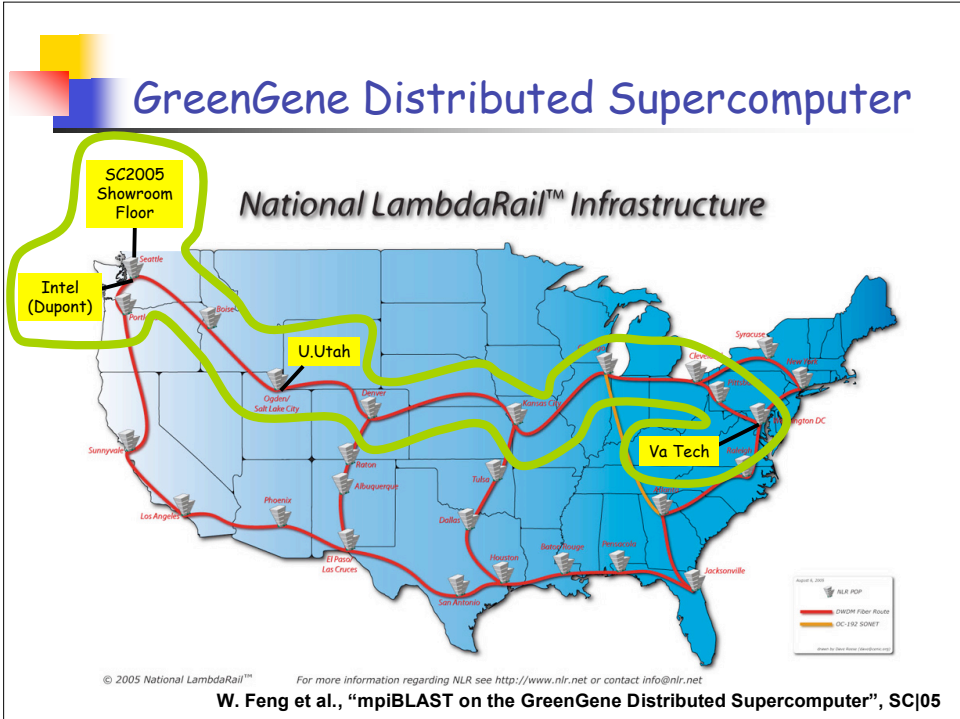
21



mpiBLAST-pio Meet The Grand Challenge: searching NT vs. NT

- SC|05 StorCloud demo (Nov. 13 - Nov. 17)
- Team
 - Institutions: LANL, NCSU, U. Utah, and Virginia Tech
 - Vendors: Intel, Panta Systems, and Foundry Networks
- Sequencing NT against itself (**16GB** raw size)
- Why?
 - Provide insightful knowledge to catalog NT database
 - Demonstrate scalability of mpiBLAST(pio) to larger problem
 - Meet the computation challenge with power of distributed parallel computing
- How?
 - GreenGene Distributed Supercomputer
 - > **3000** processors from 4 distributed sites of super computers

22





Lessons Learned from NT vs NT Search

- Results
 - Finish 526,000 sequences (1/7 NT) in one day
- Single supercomputer not enough
- Database segmentation is necessary to deal with "Hard Queries" - parallelize the computation
 - Case 1: 122k single query, take 64 procs 7 hours to finish, 1.8G output size (448hrs on single processor)
 - Case 2: 2M single query, not finished on 128 procs within 12 hours
- mpiBLAST-pio demonstrate capability of conducting large database against database sequence alignment

25



Acknowledgements

- The work of pioBLAST was funded in part or in full by the US Department of Energy's Genomes to Life program under the ORNL-PNNL project, "Exploratory Data Intensive Computing for Complex Biological Systems".
- The work of integrating data access optimizations of pioBLAST into mpiBLAST-pio was supported through Los Alamos National Laboratory contract W-7405-ENG-36.
- Other mpiBLAST-pio development contributors
Jeremy Archuleta (LANL), Avery Ching (Northwestern), Pavan Balaji (OSU)

26



References

- "Efficient Data Access for Parallel BLAST," *19th Int'l Parallel & Distributed Processing Symp.*, April 2005.
- "The Design, Implementation, and Evaluation of mpiBLAST" Best Paper: Applications Track, *4th Int'l Conf. on Linux Clusters*, Jun. 2003.
- "mpiBLAST: Delivering Super-Linear Speedup with an Open-Source Parallelization of BLAST," *Pacific Symp. on Biocomputing*, Jan. 2003.