Green Destiny + mpiBLAST = Bioinfomagic

W. Feng[a]*

[a]Research & Development in Advanced Network Technology (RADIANT),
Computer & Computational Sciences Division,
Los Alamos National Laboratory,
P.O. Box 1663, M.S. D451,
Los Alamos, NM 87545, USA

This paper outlines how our highly efficient, power-aware supercomputer called Green Destiny and our open-source parallelization of BLAST called mpiBLAST combine to create a bit of "bioinfomagic." Green Destiny, featured in The New York Times and winner of a 2003 R&D 100 Award, revolutionized high-performance computing by re-defining performance to focus on issues of efficiency, reliability, and availability. Green Destiny is a 240-processor supercomputer that operates at a peak rate of 240 billion floating-point operations per second (or 240 gigaflops) but fits in six square feet and sips as little as 3.2 kilowatts of power. Consequently, it does not require any special infrastructure to operate, i.e., no cooling, no raised floor, no air filtration, and no humidification control. These attributes resulted in interest from several pharmaceutical and bioinformatics institutions, which likewise did not have special infrastructure to house traditional supercomputing clusters.

Subsequent interactions with these pharmaceutical and bioinformatics institutions led to the birth of mpiBLAST, an open-source parallelization of BLAST that achieves super-linear speed-up via a technique called database segmentation. Database segmentation allows each computing node to search a smaller portion of the database (one that fits entirely in memory), thus eliminating disk I/O and vastly improving performance. When used in concert with Green Destiny, we demonstrate that a 300-kB BLAST query that takes nearly one full day to complete on a traditional PC or workstation takes only minutes on Green Destiny.

1. THE ORIGIN OF GREEN DESTINY & THE BIRTH OF mpiBLAST

In the summer of 2001, my research team — RADIANT: Research and Development in Advanced Network Technology — was faced with a dilemma. When running distributed network simulations on our 128-processor tranditional supercomputing cluster (i.e., Beowulf cluster [1]) during the summer heat wave of 2001 in a dusty warehouse that routinely reached 90°F (32°C), the traditional Beowulf cluster failed on a weekly basis, and oftentimes, more frequently than that. (This summer heat wave of 2001 was also the one that caused rolling brownouts in California.) These unscheduled failures easily resulted in thousands of dollars of lost productivity per week. For instance, in the typical case, system administrators would spend upwards of a half-day to diagnosis and repair problems with the cluster. And when repairs required new hardware, administrators would boot up the working portion of the cluster for use and then later shut it down for repair when the new hardware came in. From the applications standpoint, network simulations would have to be re-started. Clearly, the team could not continue operating in such an inefficient manner.

On a related note, Table 1 shows that the problem with downtime is even more insidious for business services that routinely rely on a web-server or compute-server farm (i.e., embarrassingly parallel variations of a Beowulf cluster), particularly given the fact that 65% of information technology managers report that their web sites were unavailable to customers over a six-month sampling period and that 25% of the web sites experienced three or more outages in the six-month period [2].

*This work was supported by the Los Alamos Computer Science Institute of the U.S. Department of Energy through LANL contract W-7405-ENG-36. This paper is also a Los Alamos Unclassified Technical Report: LA-UR-03-6651.

Table 1
Estimated Costs of an Hour of Server Downtime for Business Services
(Source: Contingency Planning Research, Inc.)

Service	Cost of One Hour of Downtime
Brokerage Operations	$6,450,000
Credit Card Authorization	$2,600,000
Amazon.com	$275,000
Ebay	$225,000
Package Shipping Services	$150,000
Home Shopping Channel	$139,000

To address this problem with downtime, we endeavored to research, develop, and integrate a low-power, always-available Beowulf cluster called Green Destiny [3–5] to address issues of efficiency, reliability, and availability. Green Destiny is a 240-processor supercomputer that operates at a peak rate of 240 billion floating-point operations per second (or 240 gigaflops) but fits in six square feet and sips as little as 3.2 kilowatts of power (or roughly 10 times less power than a traditional cluster). Because Green Destiny runs so cool, it does not require any special infrastructure to operate, i.e., no cooling, no raised floor, no air filtration, and no humidification control. Furthermore, in its lifetime, the cluster has never failed.

Consequently, the above attributes resulted in significant interest from several pharmaceutical and bioinformatics institutions, who likewise did not have special infrastructure to house traditional supercomputing clusters. Subsequent discussions revolved around the fact that these institutions relied heavily on a tool called "BLAST: Basic Local Alignment Sequence Tool," a ubiquitous sequence database-search tool used in molecular biology. (In general, BLAST takes a query DNA or amino acid sequence and searches for similar sequences in a database of known sequences. At the completion of the search, BLAST reports the statistical significance of the similarities between the query and the sequences in the database.) To improve the throughput of BLAST, most of these institutions used traditional Beowulf clusters in order to run multiple instantiations of the sequential BLAST code. Those that were more adventuresome looked into parallelizing the sequential BLAST code [6–12]. Ultimately, however, all the institutions have been hampered by frequent failures in their traditional Beowulf clusters (due to overheating) as well as by poor parallelizations of BLAST. These hinderances appropriately set the stage for Green Destiny and mpiBLAST, respectively.

Virtually all parallel implementations rely on a technique called query segmentation, resulting in nearly linear speed-up. A few implementations, including our own open-source implementation dubbed mpiBLAST, employ a technique that we call database segmentation [13–16]. However, our implementation leverages the ubiquitously accepted NCBI implementation of BLAST and delivers super-linear speed-up while simultaneously being free and open source. As a result, since its release in early 2003, mpiBLAST has been downloaded well over 4000 times.

The remainder of this paper presents a brief overview of BLAST, followed by details about the existing techniques to parallelize BLAST as well as the technique used in mpiBLAST, and a performance evaluation of mpiBLAST, our open-source parallelization of BLAST using the ubiquitous MPI (Message Passing Interface) [19].

2. AN OVERVIEW OF BLAST

BLAST searches a query sequence containing nucleotides (DNA) or peptides (amino acids) against a database of known nucleotide or peptide sequences. Since peptide sequences results from ribosomal translations of nucleotides, comparisons can be made between nucleotide and peptide sequences. BLAST provides the capability to compare all possible combinations of query and database sequence types by translating sequences on the fly, as shown in Table 2.

In BLAST, each search type executes in nearly the same way. The BLAST search heuristic [17]

Table 2
BLAST Search Types

Search Name	Query Type	Database Type	Translation
blastn	nucleotide	nucleotide	none
tblastn	peptide	nucleotide	database
blastx	nucleotide	peptide	query
blastp	peptide	peptide	none
tblastx	nucleotide	nucleotide	query and database

indexes both the query and target (database) sequence into words of a chosen size. It then searches for matching word pairs, called hits, with a score of at least T and extends the match along the diagonal. Gapped BLAST [18], hereafter referred to simply as BLAST, modifies the original BLAST algorithm to increase sensitivity and decrease execution time by moving down the sequences until two hits have been found, each with a score of at least T within A letters of each other. BLAST then performs an ungapped extension on the second hit to generate a high-scoring segment pair (HSP). If the HSP score is greater than a second threshold, a gapped extension is triggered simultaneously both forward and backward. The resulting output consists of a set of local gapped alignments found within each query sequence, the alignments score, an alignment of the query and database sequences, and a measure of the likelihood that the alignment is a random match between the query and database.

3. RELATED WORK

In this section, we briefly describe three approaches towards parallelizing BLAST: (1) hardware acceleration that parallelizes the comparison of a single query to a single database entry, (2) query segmentation, and (3) database segmentation.

3.1. Hardware Acceleration

Hardware accelerators parallelize the comparison of a single query sequence to a single database entry. Because symmetric multiprocessors (SMPs) and symmetric multithreaded systems (SMTs) cannot support the level of parallelism required, custom hardware must be used. Singh et al. introduced the first hardware accelerator for BLAST [12]. More recently, TimeLogic (http://www.timelogic.com) commercialized an FPGA-based accelerator called the DeCypher BLAST hardware accelerator [11].

3.2. Query Segmentation

Query segmentation provides the most natural parallelization of BLAST by splitting up a query (or set of queries) such that each compute node in a cluster (or each processor in an SMP) searches a fraction of the sequence database, as shown in Figure 1. Thus, multiple BLAST searches can execute in parallel on different queries. However, such an approach typically requires that the entire database be replicated on each compute nodes local storage system [6,7]. If the database to be searched is larger than core memory, then query-segmented searches suffer from the same adverse effects of disk I/O as in traditional BLAST. When the database fits in core memory, however, query segmentation can achieve nearly linear scaling for all BLAST search types.

3.3. Database Segmentation

Database segmentation is an orthogonal approach to query segmentation. Whereas query segmentation keeps the database intact and uses individual query segments (or sub-queries) to match against a copy of the entire database on each node, database segmentation keeps the query intact and distributes individual database segments to each node for the query to be searched upon, as shown in Figure 2. One of the biggest challenges of this approach is to ensure that the statistical scoring is properly produced as it depends on the size of the database, a database that database segmentation dutifully chops up.

Database segmentation has also been implemented in a closed-source commercial product by Tur-

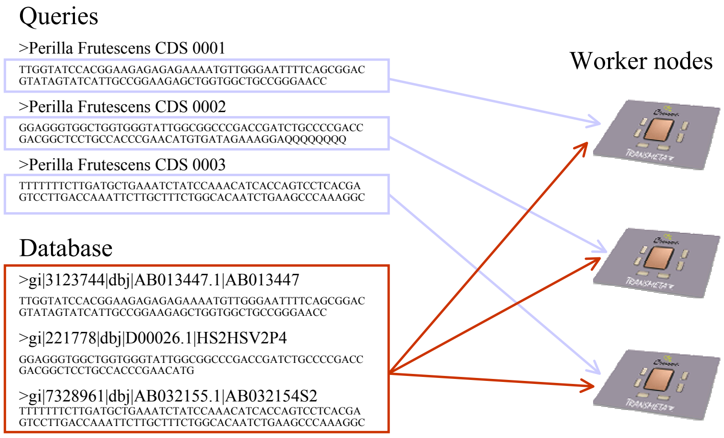

Figure 1. Query Segmentation

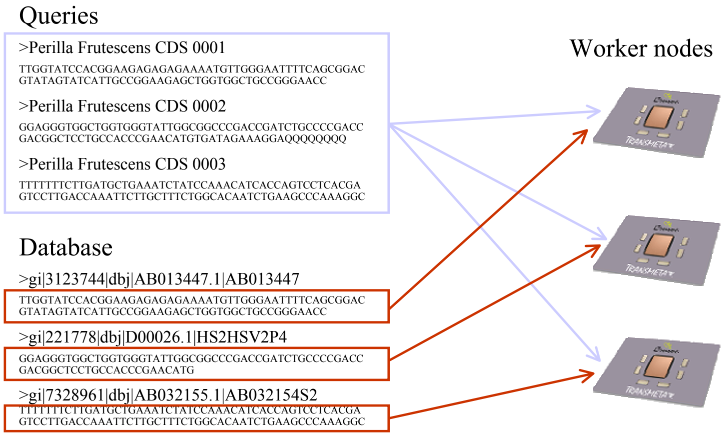

Figure 2. Database Segmentation

boWorx, Inc. called TurboBLAST [13]. However, its propietary implementation only results in linear speed-up. Recently, Mathog [16] also released an implementation of database segmentation called `parallelblast`, which is composed of a set of scripts that operate in the Sun Grid Engine (SGE)/PVM environment. Aside from requiring the SGE/PVM environment, parallelblast also differs from mpiBLAST in that it is not directly integrated with the NCBI toolkit.

4. THE mpiBLAST ALGORITHM

The mpiBLAST algorithm consists of three steps: (1) segmenting and distributing the database, e.g., see Figure 2, (2) running mpiBLAST queries on each node, and (3) merging the results from each node into a single output file [15].

The first step consists of a front-end node formatting the database via a wrapper around the standard NCBI `formatdb` called `mpiformatdb`. The `mpiformatdb` wrapper generates the appropriate command-line arguments to enable NCBI `formatdb` to format and divide the database into many small fragments of roughly equal size. When completed, the formatted database fragments are placed on shared storage.

Next, each database fragment is distributed to a distinct worker node and queried by directly executing the BLAST algorithm as implemented in the NCBI development library.

Finally, when each worker node completes searching on its fragment, it reports the results back to the front-end node who merges the results from each worker node and sorts them according to their score. Once all the results have been received, they are written to a user-specified output file using the BLAST output functions of the NCBI development library. This approach to generating merged results allows mpiBLAST to directly produce results in any format supprted by NCBI's BLAST, including XML, HTML, tab-delimited text, and ASN.1.

5. PERFORMANCE EVALUATION OF mpiBLAST

In this section, we show that mpiBLAST, with its database-segmenting technique, achieves super-linear performance gains. For instance, when increasing the number of compute nodes from 1 to 4, mpiBLAST achieves a speed-up of nearly 10, as shown in Table 3. Overall, our reference 300-KB query against the 5.1-GB uncompressed nt database takes 1346 minutes (or 22.4 hours) on one compute node and less than 8 minutes on 128 nodes of Green Destiny. (A more detailed performance analysis and evaluation can be found in [15].)

Table 3
BLAST Run Time for a 300-kB Query Against the nt Database

# Nodes	Run Time (sec)	Speed-Up
1	80775	1.00
4	8752	9.23
8	4548	17.76
16	2437	33.15
32	1350	59.83
64	851	94.92
128	474	170.41

If one looks carefully at Table 3, the efficiency of mpiBLAST decreases as the number of nodes increase. That is, if we take the "speed-up" column and divide it by the "# nodes" column, the efficiency of mpiBLAST across four nodes is 2.31 (9.23/4) and drops all the way down to 1.33 (170.41/128) when run across 128 nodes. The reason for this dropoff is due to the tradeoff that exists when segmenting the database into many small fragments. There is significant overhead in searching extra fragments; thus, the ideal database segment will typically be the largest fragment that can fit in memory and not cause any swapping to disk. Making fragments smaller than the available memory simply adds overhead [15].

6. CONCLUSION

In this paper, we briefly described how our energy-efficient and highly-reliable Green Destiny cluster led to the development of mpiBLAST to create a bit of "bioinfomagic" — e.g., moving from executing on one node to four nodes resulted in nearly a ten-fold speed-up. In short, mpiBLAST is an MPI-based [19] implementation of database segmentation for parallel BLAST searches. Its primary contributions are that it is open source, that it routinely produces super-linear speed-up, and that it directly interfaces with the NCBI development library to provide BLAST users with an interface and output formats identical to the ubiquitous NCBI-BLAST.

REFERENCES

[1] T. Sterling, D. Becker, D. Savarese, J. Dorband, U. Ranawake, and C. Packer, "Beowulf: A Parallel Workstation for Scientific Computation," Proceedings of the 24th International Conference on Parallel Processing, August 1995.

[2] T. Wilson, "The Cost of Downtime," InternetWeek, July 30, 1999.

[3] W. Feng, M. Warren, and E. Weigle, "Honey, I Shrunk the Beowulf!" Proceedings of the 31st International Conference on Parallel Processing, August 2002.

[4] W. Feng, M. Warren, and E. Weigle, "The Bladed Beowulf: A Cost-Effective Alternative to Traditional Beowulfs," Proceedings of the 4th IEEE Cluster, September 2002.

[5] M. Warren, E. Weigle, and W. Feng, "High-Density Computing: A 240-Processor Beowulf in One Cubic Meter," Proceedings of the 15th SC: High-Performance Networking and Computing Conference, November 2002.

[6] R. Braun, K. Pedretti, T. Casavant, T. Scheetz, C. Birkett, and C. Roberts, "Parallelization of Local BLAST Service on Workstation Clusters," Future Generation Computer Systems, 17(6):745-754, April 2001.

[7] N. Camp, H. Cofer, and R. Gomperts, "High-Throughput BLAST," SGI White Paper, September 1998.

[8] E. Chi, E. Shoop, J. Carlis, E. Retzel, and J. Riedl, "Efficiency of Shared-Memory Multiprocessors for a Genetic Sequence Similarity Search Algorithm," Technical Report TR97-005, Computer Science Department, University of Minnesota, 1997.

[9] E. Glemet and J. Codani, "LASSAP, a Large Scale Sequence comparison Package," Computer Applications in the Biosciences,13(2):137-143, April 1997.

[10] K. Pedretti, T. Casavant, R. Braun, T. Scheetz, C. Birkett, and C. Roberts, "Three Complementary Approaches to Parallelization of Local BLAST Service on Workstation Clusters," Lecture Notes in Computer Science, 1662:271-282, 1999.

[11] A. Shpuntof and C. Hoover, Personal Communication, August 2002.

[12] R. K. Singh, W. D. Dettloff, V. L. Chi, D. L. Hoffman, S. G. Tell, C. T. White, S. F. Altschul, and B. W. Erickson, "BioSCAN: A Dynamically Reconfigurable Systolic Array for Biosequence Analysis," Research on Integrated Systems, 1993.

[13] R. D. Bjornson, A. H. Sherman, S. B. Weston, N. Wilard, and J. Wing, "TurboBLAST: A Parallel Implemtnation of BLAST Based on the TurboHub Architecture for High Performance Bioinformatics," Proceedings of the 1st IEEE International Workshop on High-Performance Computational Biology, April 2002.

[14] A. Darling and W. Feng, "BLASTing Off with Green Destiny," IEEE Bioinformatics Conference, August 2002.

[15] A. Darling, L. Carey, and W. Feng, "The Design, Implementation, and Evaluation of mpiBLAST," ClusterWorld Conference & Expo in conjunction with the 4th International Conference on Linux Clusters: The HPC Revolution 2003, June 2003.

[16] D. R. Mathog, "Parallel BLAST on Split Databases," Bioinformatics, 19(14), 2003.

[17] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic Local Alignment Search Tool," Journal of Molecular Biology, 215:403-410, 1990.

[18] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs," Nucleic Acids Research, 25:3389-3402, 1997.

[19] W. Gropp, E. Lusk, and A. Skjellum, Using MPI, MIT Press, 1999.