Parallel Genomic Sequence-Search on a Massively Parallel System

Oystein Thorsen
IBM
Rochester, 3605 HWY 52N
Rochester, MN 55901
507-253-7915, USA
othorse@us.ibm.com

Brian Smith
IBM
Rochester, 3605 HWY 52N
Rochester, MN 55901
507-253-4717, USA
brsmith@us.ibm.com

Carlos P Sosa*
IBM and University
of Minnesota
Supercomputing Institute
117 Pleasant Street South
Minneapolis, MN 55455
612-624-2966, USA
cpsosa@us.ibm.com

Karl Jiang
IBM
Rochester, 3605 HWY 52N
Rochester, MN 55901
507-253-7915, USA
kjiang@us.ibm.com

Heshan Lin
North Carolina State University
Department of Computer Science
3226 EB II
Raleigh, NC 27695
919-513-7577, USA
hlin2@ncsu.edu

Amanda Peters
IBM
Rochester, 3605 HWY 52 N
Rochester, MN 55901
507-253-7008, USA
apeters@us.ibm.com

Wu-chun Feng
Virginia Tech
Department of Computer Science
2202 Kraft Drive, 209 KWII
Blacksburg, VA 24060
540-231-1192, USA
feng@cs.vt.edu

ABSTRACT

In the life sciences, genomic databases for sequence search have been growing exponentially in size. As a result, faster sequence-search algorithms to search these databases continue to evolve to cope with algorithmic time complexity. The ubiquitous tool for such search is the Basic Local Alignment Search Tool (BLAST) [1] from the National Center for Biotechnology Information (NCBI). Despite continued algorithmic improvements in BLAST, it cannot keep up with the rate at which the database is exponentially increasing in size. Therefore, parallel implementations such as mpiBLAST have emerged to address this problem. The performance of such implementations depends on a myriad of factors including algorithmic, architectural, and mapping of the algorithm to the architecture. This paper describes modifications and extensions to a parallel and distributed-memory version of BLAST called mpiBLAST-PIO and how it maps to a massively parallel system, specifically IBM Blue Gene/L (BG/L). The extensions include a virtual file manager, a "multiple master" runtime model, efficient fragment distribution, and intelligent load balancing. In this study, we have shown that our optimized mpiBLAST-PIO on BG/L using a query with 28014 sequences and the NR and NT databases scales to 8192 nodes (two cores per node). The cases tested here are well suited for a massively parallel system.

* Corresponding author.

CF'07, May 7–9, 2007, Ischia, Italy.

Categories and Subject Descriptors

K.2 [**IBM**]: Blue Gene/L; H.4 [**Information Systems Applications**]: Bioinformatics; D.2.8 [**Software Engineering**]: Metrics - performance measures.

General Terms

Algorithms, Measurement, Performance, Theory.

Keywords

mpiBLAST-PIO, massively parallel computer, Blue Gene/L, multiple masters parallelization.

1. INTRODUCTION

The human genome project was completed in 2003. Two of the goals of the project involved determining the sequences of the three-billion chemical base pairs in DNA and storing this information in databases. Nowadays, scientists continue sequencing other organisms and storing all this information in databases. Databases are currently growing exponentially. This growth has prompted programmers to develop faster and more sophisticated algorithms to keep pace with the increasing sizes of the databases. Software improvements combined with state-of-the-art hardware will prove to play a vital role in the future of computational biology. This is particularly true when it comes to being able to extract and analyze data accumulated in these databases.

The list of molecular biology databases is constantly increasing and more scientists rely on this information [2]. The Nucleic Acids Research (NAR) Molecular Biology Database collection reported an increase of 139 more databases for 2006 compared to the previous year. GenBank doubles its size approximately every 18 months [3]. However, the increase in microprocessors clock

speed is not changing at the same rate [4]. Therefore, scientists try to leverage the use of multiple processors.

One of the most popular programs to search databases is BLAST [1,5]. It is extensively used worldwide and its popularity is due to its powerful, flexible and fast implementation. BLAST relies on a heuristic algorithm to identify local alignments [1].

The demand for programs that can help analyze all this data will continue to increase. Carrying out, for example, database searches and sequence alignment tend to be not only computationally intensive but I/O demanding [6,7]. This becomes apparent as the size of the query and database increase [4,6,7]. Single BLAST searches can be performed in a matter of seconds or minutes. However, a recent study carried out a search of the NT database with the same NT database as a query have illustrated how much of a daunting task this may turn out to be [6,7]. Clearly, as data continues to grow, so does the need for software optimized for highly parallel systems that can leverage thousands of processors either on a grid [7] or on a massively parallel system [8].

2. Other Related Work

2.1 Shared-Memory Parallelization

To perform database searches in a shared-memory environment, BLAST carries out memory-mapped I/O via `mmap` [9]. The advantage of memory-mapped I/O is that a file on disk gets mapped into a buffer in memory that corresponds to bytes in the file [9]. Symmetric multi-processor (SMP) machines with larger memories can fully utilize all the available memory to store the database.

In this scheme, each query sequence is compared against the entire database shared in memory. This step can be parallelized using multi-threading on an SMP. The parallel BLAST algorithm divides the database in sections that are allocated to different threads [4,10]. Given the amount of memory on a local node on a massively parallel machine, a different approach is required. Especially since `mmap` is not supported as part of the BG/L operating system [11].

2.2 Distributed-Memory Parallelization

Two of the authors in this study have previously shown that one of the key ideas in parallelizing BLAST on distributed-memory machines relies on database fragmentation (or pre-distribution) [6,7]. Depending on the parallelization scheme, different fragments of the query will search the entire database on different nodes or the entire query will search fragments of the database on different nodes [12]. This process may be implemented within three levels of parallelism. Fine grained, medium grained, coarse grained [6,7,12]. Each level of parallelism might be better suited for a particular architecture [12].

The need for running BLAST on distributed memory machines led to the creation of mpiBLAST and subsequent versions [13,14]. This distributed-memory version uses the message passing interface (MPI) [15] and the parallelization is based on a master/worker scheme as described in refs. [6,7,13,14]. This approach has been divided into a "master-writing" implementation where the master is the focal point and coordinates not only all the information received from the workers but I/O as well [6]. Although this approach has certain advantages, it is not well suited for massively parallel machines since the master becomes the bottleneck as more nodes are added to finish the task sooner [8].

The second approach is to let the workers carry out their own I/O. There are two ways to perform this task: the "worker-writing" approach via collective I/O or individual I/O [6]. The latter has been shown to be more efficient [6].

A recent study has reported on an implementation called ScalaBLAST. This new parallelization makes use of the Global Arrays toolkit [16] to distribute the database to remote nodes on the system, combined with a "worker-writing" scheme [17].

Other approaches include specialized hardware for sequence alignment [18,19] as well as a high-throughput approach via a central client that parses sequences to run in parallel [20,21]

2.3 Our Contribution

In this study, we describe our efforts in optimizing mpiBLAST-PIO for a massively parallel system. Our approach relies on eliminating I/O by shifting the location of the database from disk to compute node memory. We also implement the multiple masters scheme previously reported [8]. The rest of the paper is organized as follows: the next section describes the hardware and software that was utilized in this study. Next, we describe our parallel implementation for massively parallel machines. Performance results and discussion provide information about scalability and relates the performance to some of the parallelization techniques utilized in this work. Finally we present a conclusions section.

3. System Overview

3.1 Hardware

We have previously described BG/L in detail [11]. However, some of the architectural features are relevant for this study. We briefly summarize some of the key components of BG/L in this section, which are shown in Fig. 1. The smallest component is the *chip*. This first component is illustrated in Figure 1. The BG/L basic block is a PowerPC 440 dual-core processor (one node). Each processor core runs at a frequency of 700 MHz and each processor core can perform four floating-point operations per cycle, giving a theoretical peak performance of 5.6 Gflops/chip. The chip constitutes the compute node [11]. *This is what we refer to in this work as nodes.*

Next is the *compute card* shown in Figure 1. Two compute nodes attached to a processor card along with memory (RAM) create a compute card (two nodes). The amount of RAM per card is 2 GB (1 GB per compute node) [11]. The *I/O card* is next. This card is physically very similar to the compute card, however, the I/O card has integrated Ethernet enabled for communication with the outside world [11]. *The I/O cards and compute cards are all plugged into a node card or node board.* There are two rows of eight compute cards on the node card and 0, 1, or 2 I/O cards depending on the I/O configuration [11]. A *midplane* consists of 16 compute cards stacked in a rack. A *rack* holds two midplanes, for a total of 32 node cards or 1024 compute nodes illustrated in Figure 1. The largest system currently produced is made of 64 racks for a total of 65536 compute nodes [11].

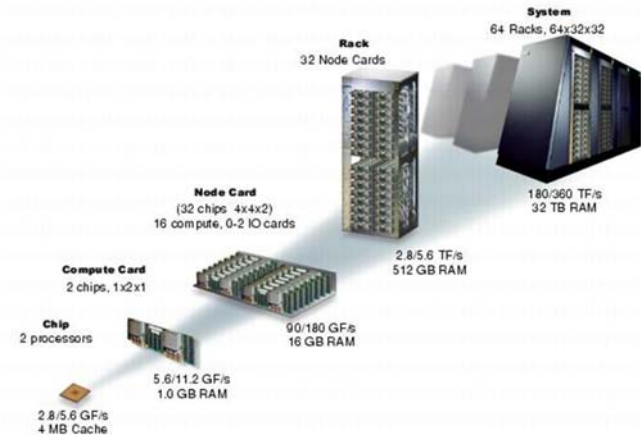

Figure 1. Copyright IBM Corp. 2004. Blue Gene/L components. Courtesy of International Business Machines Corporation. Unauthorized use not permitted. Image obtained from Ref. [11].

Finally, the compute nodes may be configured at boot time in one of following ways: the first way is the virtual node mode (VN). This configuration uses both processors separately, running a different process of the user's application on each processor with half the RAM assigned to each processor. The second way is Co-processor node mode (CO). This configuration uses the secondary processor as an offload coprocessor for processing the I/O of the main processor [11]. In this study, we only used the CO mode to take advantage of all the physical memory available per node.

3.2 Software

mpiBLAST is an open-source parallelization of BLAST that uses MPI [13]. One of the key features of the initial parallelization of mpiBLAST is its ability to fragment and distribute databases.

In this study, we use the mpiBLAST 1.4-PIO program [13,14]. This version incorporates most of the functionality introduced in pioBLAST [14]. This version can perform parallel I/O via MPI I/O. It also introduces the use of "workers-write" when parallel files systems are available and the "master-write" scheme when file systems are NFS-mounted.

4. Massively Parallel Implementation

4.1 Storing Files in Memory

As previously pointed out, the original BLAST version utilizes `mmap` to store the database in memory. Since `mmap` is not implemented as a part of the BG/L operating system, the application falls back on reading files directly from disk. With all nodes sharing the same file system, I/O contention severely limits the scaling of this application. To address this, we implemented a modified version of the "virtual file manager" (VFM) that was successfully used in pioBLAST which is described in ref. [14]. VFM is not only used to store the database fragments in memory, but also the query file and various temporary files. This eliminates disk I/O and allows file distribution using MPI when workers need the same file. Fragment distribution and query distribution are described in detail in sections 4.3 and 4.4.

In BLAST and mpiBLAST-PIO, there are two structures that are central when dealing with virtual files. The one most specific to virtual files is `Nlm_MemMap` which contains the actual pointer to the memory location where the file is stored in memory. The second one is `NlmMFILE` which contains the `Nlm_MemMap` object, but could also contain a file pointer if we wanted to read directly from the file.

The `Nlm_MemMap` object is initialized by `Nlm_MemMapInit` which normally would call `mmap` or return a file pointer. In the case of VFM we had to make this function allocate enough space to contain the entire file and copy the contents of the file into memory before closing the file.

The `NlmMFILE` object is contained within VFM and is initialized the first time the program wants to access a file. Otherwise VFM will only return a pointer to the `NlmMFILE` object already in memory.

4.2 Multiple Masters

The biggest problem with scalability in the original implementation was that there was a lot of administration work put on the master to make the workers as efficient as possible. Since this work increased linearly with the number of workers, it would get overwhelmed and cause the workers to wait longer and longer for a response every time they wanted more work. This is pictorially described in Figure 2.

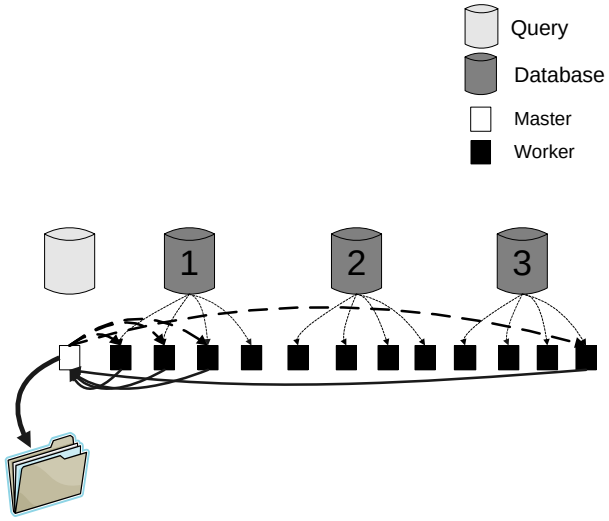

Figure 1. Original implementation. "Single master to many worker nodes" communication.

By introducing a second level of management, we could limit the number of workers for the master by creating groups of nodes containing one master for each group working on separate query sequences. So instead of doubling the number of workers, we double the number of groups whenever we doubled the number of total nodes. This ensured that the number of master nodes would scale with the number of total nodes. Currently the second level of management only contains one node (the SuperMaster) and could potentially be a bottleneck, but this node does very little work (described in section 4.5), and our benchmarks show that it can handle scalability to 8192 nodes. Figure 3 provides an overview of the steps involved in the new scheme.

The amount of work required by the master depends on multiple factors and could change depending on what query and/or database is used. When we initially tested the original version, it only scaled to 64 nodes in co-processor mode. This is similar to what has been previously reported in the literature [17]. So we selected 64 as a good group size. Since a query sequence is not split between groups, each group must contain at least one copy of each database fragment. Each group must therefore have at least enough aggregate memory to contain the entire database and each node must have at least enough memory to contain one database fragment. Figure 3a shows the creation of groups.

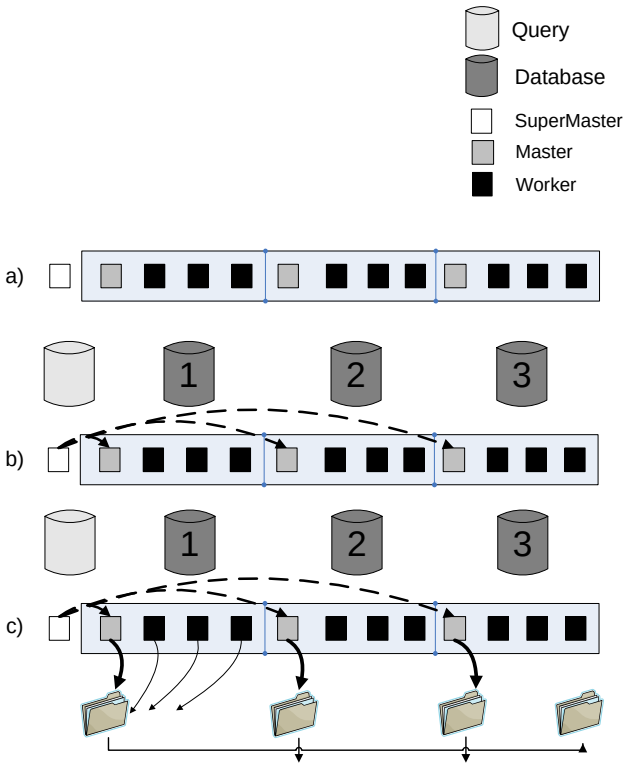

Figure 3. Massively parallel implementation. a) Divide into groups, b) query fragmentation and load balancing, c) multiple outputs

During the setup phase, the master carries out calculations such as the E-value [1] which is now divided among all masters to shorten the setup phase. All masters split the work between them before gathering the results and distributing it among all workers (Figs. 3b and 3c).

4.3 Fragment Distribution

When using very few nodes, it will be faster for all of them to read their database fragments directly from disk, but due to the limited scaling of disk I/O bandwidth it will be much faster to let one node per fragment read from disk and distribute to everyone else who needs the fragment. A regular broadcast is highly optimized for speed and scalability in all MPI implementations making this new method a superior choice even on very few groups.

To maximize the use of available bandwidth for the initial read, the nodes which read from disk are spread out over all the groups (and I/O nodes). This is illustrated in Figure 3b. And if each node has more than one fragment we make sure that a node does not have to read more than one fragment from disk to allow all reads from disk to be performed concurrently.

The whole scheme is made possible with the addition of fragment communicators. For every fragment that is to be distributed, a communicator is constructed containing every node that is to receive that fragment. Calculations are made that ensure that if there are nodes that will receive more than one fragment, each node is assigned to read only a particular fragment. Also, instead of keeping a record in a file of what local fragments the node has on disk, the node instead keeps a virtual record of fragments the node is currently storing in memory. This list is broadcasted to the scheduler for scheduling purposes.

Finally, there is a parameter that represents the minimum number of times the database is duplicated over a particular group. For some query and database combinations, the number of sequences in each fragment being unevenly distributed will cause some nodes to be far ahead of the others in terms of the query sequence they are working on. By having more than one fragment on a node, this imbalance is less likely. However, in most cases this parameter has negligible effect.

4.4 Query Distribution

In mpiBLAST-PIO, the master node reads the query file, counts the number of sequences and distributes them to the workers and then writes to a temporary file for each worker. In our implementation, the query file is stored as a virtual file instead of being written to file. Sequences are loaded from this virtual file instead of directly from disk. This is done using sequence loading functions within the NCBI toolkit that accepts a `char` pointer instead of a `file` pointer.

4.5 Load Balancing

In general, different query sequences are normally not processed in the same amount of time due to the heuristic nature of the search algorithm and variations in sequence length. So even if the groups are given the same amount of query sequences to work on, their total run time is usually quite different. It was therefore important to be able to hold back on some of the query sequences for the groups that finish early. To do this we used a dedicated node referred to as the SuperMaster.

The SuperMaster's only job is to make sure that none of the groups finish earlier than the rest. Our benchmarks show that the best way of doing this is to distribute half of the query sequences at the beginning, and then whenever a group finishes, the SuperMaster would give it a number of new query sequences equaling the number of remaining queries divided by the number of groups. This way the SuperMaster will give out fewer and fewer query sequences every time. If there are no more query sequences left, the SuperMaster will tell the group to finish up and quit the job.

In many cases, one of the groups will be smaller than the rest (if the total number of nodes, minus one for the SuperMaster, is not a multiple of the group size) so the assigned query sequences assigned to this group must be reduced proportionally to its size.

The rationale behind this is that the worker should at least manage to do its part of the remaining queries. Even if the new query sequences would require more work than the other sequences, it already had a head start since it was the first group to finish with its previous batch.

This scheme results in more and more work for the SuperMaster. In order to prevent the groups from spending too much time waiting for the SuperMaster at the end, we set a minimum number of queries to distribute. The ideal number for this is also dependent on the query and database. The more work required for each query sequence, the smaller this number should be, and the more groups there are, the larger this number should be. Our benchmarks show that *five* is a good number for all the tests we have done in this paper.

5. Performance Results

5.1 Queries and Databases

The query used in this study corresponds to *Arabidopsis thaliana*. This is a model organism for studying plant genetics. This query was further subdivided into small, medium and large query sets containing 200, 1168 and 28014 sequences, respectively. This set was obtained from a previous study [22,23].

We used two databases to carry out all the performance experiments. These databases are maintained by NCBI [24]. The first database is the non-redundant (NR) protein database. The version that we downloaded contains 2,451,907 sequences with a total of 830,525,235 letters. The second database is the NT database. The version that we downloaded has 4,218,264 sequences with a total of 17,671,372,779 letters.

5.2 Results

The evaluation of the massively parallel version of mpiBLAST-PIO was carried out using two of the programs available as part of the BLAST suite of programs. The first program that we tested is blastx. This program uses an algorithm to compare the six-frame translation of our query sets (nucleotides) against the protein NR database. The second program that we tested corresponds to blastn. This program uses an algorithm to compare a nucleotide query sequence against a nucleotide database – in our case, the NT database.

Fig. 4 illustrates the results of comparing three queries of three different sizes on the NR database. We label the query sizes as small, medium, and large. This figure shows that scalability is a function of the query size. The small query scales to approximately 1024 nodes in co-processor mode with a parallel efficiency of 72% were the large query scales to 8192 nodes with a parallel efficiency of 74%. We did not have the opportunity to test beyond 8192 nodes for this database.

Table 1 summarizes the total execution time in seconds for 32 to 8192 nodes, which is the basis for Fig. 4. All the runs were carried out in co-processor mode. All benchmarks were run with a group size of 64. The NR database was fragmented into 31 fragments.

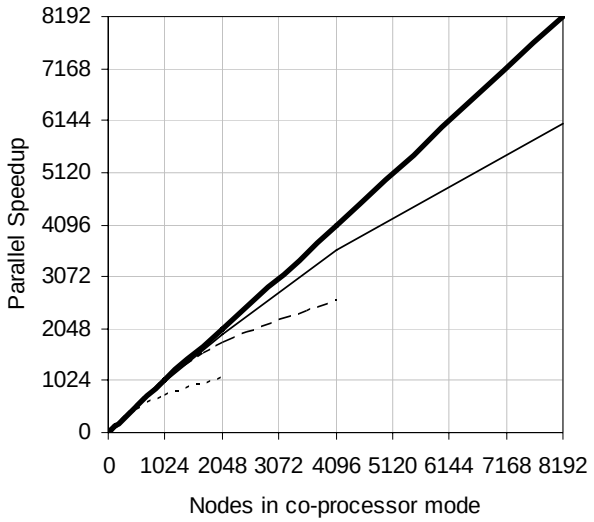

Figure 4. Scaling chart for queries run versus the NR database. From the top, the thick solid line corresponds to ideal scaling; the thin solid line corresponds to the large query; the dashed line corresponds to the medium query; the dotted line corresponds to the small query.

Table 1. Total execution times for queries run versus the NR database.

Nodes[a]	Small[b]	Medium[b]	Large[b]
32	721.8	4073.9	
64	334.1	1963.9	
128	171.7	993.7	
256	96.2	504.0	
512	50.0	251.4	5620.6
1024	31.5	131.2	2863.0
2048	21.3	73.4	1484.6
4096		50.3	796.7
8192			479.2

[a] All runs in co-processor mode

[b] All timings in seconds

Fig. 5 shows the results of comparing the same three queries but on the NT database, which is several orders of magnitude larger than the NR database. This plot also illustrates similar results as in the case of Fig. 4. Scalability for the small query is not very high, almost linear scaling up to 128 nodes (87%) in co-processor mode and continues to slightly scale up to about 1024 nodes where the parallel efficiency drops to 22%. The medium query shows a parallel efficiency of 61% at 1024 nodes. On the other hand, the large query shows almost linear scaling up to 2048 nodes in co-processor mode (93%) and continues to scale to 8192 nodes (70%) and 16384 (55%).

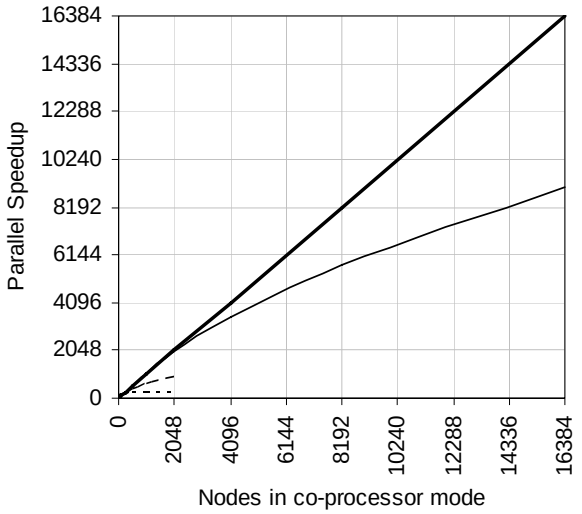

Figure 5. Scaling chart for queries run versus the NT database. From the top, the thick solid line corresponds to ideal scaling; the thin line corresponds to the large query; the dashed line corresponds to the medium query; the dotted line corresponds to the small query.

Table 2 summarizes the total execution time in seconds for 64 to 16384 nodes which is the basis for Fig. 5. All the runs were carried out in co-processor mode. All benchmarks were run with a group size of 64. The NT database was divided into 127 fragments.

Table 2. Total execution time for queries run versus the NT database.

Nodes[a]	Small[b]	Medium[b]	Large[b]
64	435.8	2395.2	
128	250.6	1374.9	
256	193.7	690.2	
512	142.9	416.1	7108.0
1024	125.3	244.2	3589.6
2048	111.6	178.4	1895.7
4096			1037.1
8192			636.3
16384			402.9

6. DISCUSSION

The original version of mpiBLAST-PIO only scaled to approximately 64 nodes in co-processor mode on BG/L. This is consistent with previous studies of mpiBLAST on BG/L [22,23]. This was due to a combination of limitations of dividing the database, the number of files that could be open at the same time, and the number of compute nodes on BG/L [22,23]. In order to overcome these limitations we needed to implement functionality available as part of pioBLAST, but not yet implemented as part of mpiBLAST-PIO. This has been shown to make a significant difference in terms of performance on a system like BG/L [22].

The first step to achieve scalability required the elimination of I/O. This was accomplished via the VFM. In previous studies, different attempts have been tried to improve I/O performance [22]. These included a server on a remote system (front-end) to read databases and then distribute them to the BG/L nodes [22,23] and the use of multiple identical file systems as an alternate method to improve I/O [22]. In our work, the database distribution is carried out by having only one worker reading a fragment of the database and then distributing it via broadcast. This type of communication is highly optimized on BG/L [11]. In addition, the query is read by the master and distributed as well. This approach showed to be efficient for the cases tested in this study.

The use of multiple masters is also proved to be important to be able to scale almost linearly (when compared to ideal scaling) for the queries and databases (medium and large) tested here. Also, key to the multiple masters scheme was the selection of the size of the groups. The worker-to-master ratio was set to 63 (64 nodes in a group) as this was tested to be the size at which optimum performance was achieved. Fig. 6 illustrates the effect of group sizes using the medium query and the NR database. The peak or optimal performance occurs with a group size of 64. The pattern observed in Fig. 6 is dependent on multiple factors that play a role when group size changes. First is the number of masters, when the group size is 64, there are precisely 8 masters. At a group size of 63 the last master has a group of only 7. Though attempts were made to make sure each group gets a fair amount of work, it is desirable that all groups have the same size.

Another factor occurs during the setup phase. Adding more masters means more nodes working on the E-value calculations. However, while each additional master speeds up the setup phase, it reduces the number of total workers. In many cases for small queries on large databases, the setup phase will dominate the total run time, and it would be beneficial to have more masters; but otherwise, the optimal group size is dependent on how many workers can handle in the main phase.

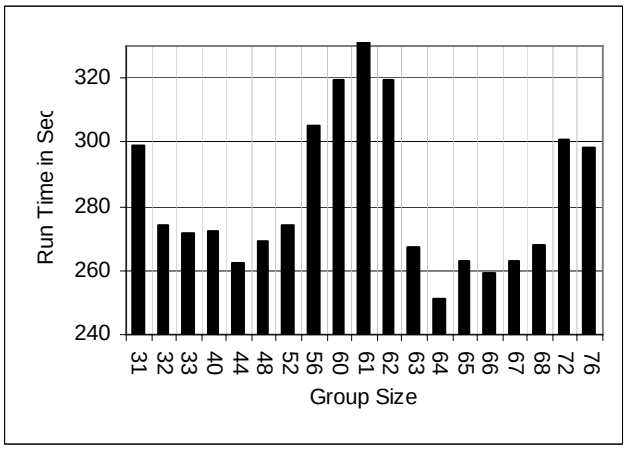

Figure 6. Run time as a function of group sizes using 512 nodes in co-processor mode with the medium query and NR database.

Load balancing is important as well since due to the heuristic nature of BLAST not all searches using different sequences take an equal amount of time. This is where the SuperMaster helped improve scalability. The SuperMaster oversees the work being carried out by all the groups and makes sure that there are no idle groups for long periods of time, hence improving load balance.

What percentage of the query is initially distributed (as long as the percentage is not too large, 50% is certainly reasonable) or how many queries are given at a time in the beginning is not very important. We found that distributing either 1000 queries, 2000 queries, or 3000 queries at the beginning had no effect on the overall run time. The minimum distribution number, however, did have an effect (See Fig. 7).

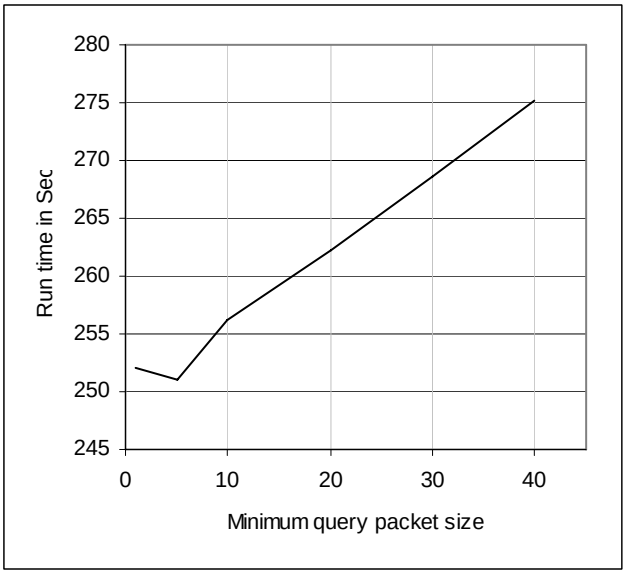

Figure 7. Performance as a function of minimum query packet size distribution.

Towards the end of the run, generally the fewer queries distributed the better (to a reasonable limit). If the query distribution amount is unreasonably low, the SuperMaster will get flooded with requests, but if it is too high, the queries will not be effectively load balanced.

A combination of eliminating I/O, proper database distribution, query distribution, multiple masters, and load balancing shows that our improvements combined with the BG/L architecture allows very high throughput on at least 4096 nodes. When the number of query sequences drops, there will be fewer and fewer jobs to distribute resulting in lower efficiency, as shown in Figs. 4 and 5.

7. CONCLUSIONS

This paper proposes a parallel scheme to increase scalability of the popular mpiBLAST-PIO program. In this work we have shown that our new implementation shows good scalability on a massively parallel system. mpiBLAST-PIO now scales to thousands of processors for the cases tested here. We report several levels of optimization, starting with storing files in memory, the introduction of a multiple masters configuration, fragment distribution and load balancing. With this new implementation, we have demonstrated that for the cases tested here, mpiBLAST-PIO is well suited for a system with thousands of distributed compute nodes. The scalability is nearly linear for all the hardware configurations tested in this study.

8. ACKNOWLEDGMENTS

We would like to thank Carl Obert for his support and valuable discussions on this work. We also would like to thank the Extreme Blue and Speed Team programs at IBM Rochester Minnesota, sponsors of this project. We especially would like to thank the staff members at the OnDemandCenter, Cindy Mestad, Steven M Westerbeck, and Geoffrey S Costigan and the Deep Computing Institute at IBM Watson, Fred Mintzer for providing valuable resources and assistance throughout the entirety of this project. We also would like to thank Huzefa Rangwala for his assistance and sharing with us his pioBLAST work on Blue Gene/L.

9. REFERENCES

[1] Altschul, S., Gish, W., Miller, M., Myers, E., and Lipman, D. J. “Basic Local alignment Search Tool,” *Journal of Molecular Biology*, vol. 215, pp. 403-410, 1990.

[2] Galperin, M. Y. “The Molecular Biology Database Collection: Update,” *Nucleic Acids Research*, vol. 34, pp. D3-D5, 2006.

[3] Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., and Wheeler, D. L. “GenBank,” *Nucleic Acids Research*, vol. 34, pp. D16-D20, 2006.

[4] Huai-hsin Chi, E., Shoop, E., Carlis, J., Retzel, E., and Riedl, J. “Efficiency of Shared-Memory Multiprocessors for a Genetic Sequence Similarity Search Algorithm,” 1997. http://www-users.cs.umn.edu/~echi/papers/perf/perf.html.

[5] Altschul,, S., Madden, T. L., Schaffer, A. A., Zheng, J., Zhang, Z., Miller, M., Lipman, D. J. “Gapped BLAST and PSI-BLAST. A New Generation of Protein Database Search Programs,” *Nucleic Acid Research*, vol. 25, pp. 3389-3402, 1997.

[6] Ching, A., Feng, W. –C., Lin, H., Ma, Y., Chodhary, A. “Exploring I/O for Parallel Sequence-Search Tools with S3aSim,” *15th IEEE International Symposium on High Performance Distributed Computing*, pp. 229-240, 2006.

[7] Gardner, M. K., Feng, W. –C., Archuleta, J., Lin, H, Ma, X., “Parallel Genomic Sequence-Searching on an Ad-Hoc Grid: Experiences, Lessons Learned, and Implications,” *Supercomputing 2006*, Florida.

[8] Jiang, K., Thorse, O, Peters, A., Smith, B., and Sosa, C. P., “An Efficient Parallel Implementation of the Hidden Markov Methods on a Massively Parallel System,” *IEEE Transactions on Parallel and Distributed Systems* 2007, accepted..

[9] Sosa, C.P., Accapadi, M., and Atyam, B. V. “BLAST Throughput Benchmarks: mmap versus read,” *in Performance Marketing Council 2003*, Ed. Stahl, E., *REDP-3692-00*, IBM Corporation, International Technical Support Organization, Austin, TX, 2003. http://publib-

b.boulder.ibm.com/Redbooks.nsf/RedpaperAbstracts/redp3692.html?Open

[10] Sosa, C. P., Tu, Z. J., and Fast, P. L. "Some Practical Suggestions for Performing NCBI BLAST Benchmarks on a pSeries 690 System," *REDP-0437-00*, IBM Corporation, International Technical Support Organization Austin, TX, 2002. http://www.redbooks.ibm.com/abstracts/redp0437.html?Open

[11] Lascu, O., Allsopp, N., Vezolle, P., Follows, J., Hennecke, M., Ishibashi, F., Paolini, M., Prakash, S., Reddy, H., Sosa, C. P., Tabary, A., Quintero, D. "Unfolding the IBM Server Blue Gene Solution," *SG24-6686-00*, IBM Corporation, International Technical Support Organization, Poughkeepsie, NY, 2005, http://www.redbooks.ibm.com/abstracts/sg246686.html?Open

[12] Braun, R. C., Pedretti, K. T., Casavart, T. L., Sheetz, T. E., Birkett, C. L., and Roberts, C. A. "Parallelization of Local BLAST Service on Workstation Clusters," *Future Generation Computer Systems*, vol. 17, pp. 745-754, 2001.

[13] Darling, A., Carey, L., and Feng, W.-C., "The Design, Implementation, and Evaluation of mpiBLAST," *Proc 4th International Conference on Linux Clusters* in conjunction with *ClusterWorld*, 2003.

[14] Lin, H., Ma, X., Chandramohan, P., Geist, A., and Samatova, N., "*Efficient Data Access for Parallel BLAST*," *Proc. 19th International Parallel and Distributed Processing Symp. (IPDPS)*, 2005.

[15] Message Passing Interface Forum. MPI: Message-Passing Interface Standard, June 1995.

[16] Nieplocha, J., Harrison, R., and Littlefield, R. "Global Arrays: A Nonuniform Memory Access Programming Model for High-Performance Computing," *J. Supercomputing* vol. 10, pp. 197-220, 1996.

[17] Oehmen, C. and J. Nieplocha, "ScalaBLAST: A Scalable Implementation of BLAST for High-Performance Data-Intensive Bioinformatics Analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, pp. 740-749, 2006.

[18] Meng, X. and Chaudhary, V. "Bio-Sequence Analysis with Cradle's 3SoCTM Software Scalable System on Chip," *Proc. ACM Symp. Applied Computing*, 2004.

[19] Muriki, K., Underwood, K., and Sass, R. "RC-BLAST: Towards a Portable, Cost-Effective Open Source Hardware Implementation," *Proc. HICOMB 2005, 4th IEEE International Workshop on High-Performance Computational Biology*, 2005.

[20] Bjornson, R., Sherman, A., Weston, S., Willard, N., and Wing, J., "TurboBLAST: A Parallel Implementation of BLAST Built on the TurboHub," *Proc. 16th International Parallel and Distributed Processing Symp. (IPDPS)*, 2002.

[21] Camp, N., Cofer, H., and Gomperts, R., "High-Throughput BLAST," 1998.

[22] Bachega, L.; Lantz, E., Moore, N, Rangwala, H., "Life Science Application Analysis Speed Team," *Final Report*, IBM, Rochester, MN, September 2005.

[23] Rangwala, H., Lantz, E., Musselman, R., Pinnow, K., Smith, B., and Wallenfelt, B. "Massively Parallel BLAST for the Blue Gene/L," *High Availability and Performance Workshop*, 2005. http://xcr.cenit.latech.edu/hapcw2005/papers/final_bglBLAST.pdf.

[24] Standard set of BLAST databases for Nucleotide, Protein, and Translated BLAST searches are made available at: ftp://ftp.ncbi.nih.gov/blast/db/ in pre-formatted format. The FASTA databases reside under the /blast/db/FASTA directory.

10. BIOGRAPHY

Oystein Thorsen is a Ph.D. student in the Department of Computer Science at Michigan Technological University. He received a master's degree in Computer Science from Michigan Technological University in 2006 and a Bachelor's degree in Computer Engineering from Agder University College, Norway, in 1999. During this time he has also worked as a Research Assistant, working on various topics related to high-performance computing and Unified Parallel C. He is currently part of the co-op program at IBM Rochester working on Life Science applications for the Blue Gene Software development team.

Brian Smith received a M.S. degree in Computer Engineering from Iowa State University in January 2005. He received B.S. degrees in both computer engineering and electrical engineering from Iowa State University in 2000. He currently works at IBM in Rochester, MN on the Blue Gene supercomputers. His job responsibilities include the communications stack for Blue Gene and applications porting and optimizing. He previously worked on high-performance computing at the US DOE Ames Laboratory Research lab. He is a member of the IEEE.

Carlos P Sosa is a Senior Technical Staff Member in the Systems and Technology Group of IBM and the technical lead of the Chemistry and Life Sciences team in the Blue Gene/L development group. His work is on scientific applications with emphasis in Life Sciences, parallel programming, benchmarking, and performance tuning. He has authored and coauthored multiple peer-reviewed papers. He also coauthored two IBM RedBooks: Unfolding the IBM eServer Blue Gene Solution and Advanced POWER Virtualization on IBM eServer p5 Servers: Architecture and Performance Considerations. He received a Ph.D. degree in Physical Chemistry from Wayne State University and completed post-doctoral work at the Pacific Northwest National Laboratory. His research areas of interest are in future POWER architectures, massively parallel computing and cellular molecular biology. He is a member of the IEEE Society, American Chemical Society, and International Society for Computational Biology.

Karl Jiang is currently an undergraduate at the University of Miami studying computer engineering, physics, and mathematics, and is currently working as part of the a co-op program at IBM Rochester. He works on parallelizing Life Sciences applications on the Blue Gene massively parallel supercomputer.

Heshan Lin is currently a Ph.D. student of Department of Computer Science at North Carolina State University (NCSU). His research mainly focuses on high-performance computing, parallel I/O and distributed computing. He received a Bachelors of Arts in Applied Mathematics from South China University of Technology in 1998, and Masters of Science in Computer Science

from Temple University in 2004. Heshan was a summer intern at Oak Ridge National Laboratory in 2004 and Los Alamos National Laboratory in 2005. Heshan has several publications in ACM and IEEE-sponsored conference proceedings. The optimizations based on one of his research projects have been incorporated into mpiBLAST-PIO, a widely used open-source bioinformatics tool. He is also the major developer of the mpiBLAST-PIO open-source project.

Amanda Peters received B.S. degrees in both computer science and physics from Duke University in 2005. She currently works at IBM in Rochester, MN on the Blue Gene supercomputers. She is involved with the porting, validating, and optimizing of Life Science applications.

Wu-chun Feng recently joined Virginia Tech as an Associate Professor of Computer Science and Electrical & Computer Engineering. Previous professional stints include Los Alamos Naitonal Laboratory, The Ohio State University, Purdue University, University of Illinois at Urbana-Champaign, Orion Multisystems, Vosaic, NASA Ames Research Center, and IBM T.J. Watson Research Center.

His research interests encompass high-performance networking and computing, low-power and power-aware computing, high-speed monitoring and measurement, and bioinformatics. He is the author or co-author of over a hundred peer-reviewed technical publications in the above areas.

Dr. Feng received B.S. degrees in Computer Engineering and Music and a M.S. degree in Computer Engineering from Penn State University in 1988 and 1990, respectively. He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1996. He is a Senior Member of IEEE