

Distributed Sequence Alignment Applications for the Public Computing Architecture

Stephen Pellicer*, Guihai Chen, *Member, IEEE*, Keith C. C. Chan, and Yi Pan, *Senior Member, IEEE*

Abstract—The public computer architecture shows promise as a platform for solving fundamental problems in bioinformatics such as global gene sequence alignment and data mining with tools such as the basic local alignment search tool (BLAST). Our implementation of these two problems on the Berkeley open infrastructure for network computing (BOINC) platform demonstrates a runtime reduction factor of 1.15 for sequence alignment and 16.76 for BLAST. While the runtime reduction factor of the global gene sequence alignment application is modest, this value is based on a theoretical sequential runtime extrapolated from the calculation of a smaller problem. Because this runtime is extrapolated from running the calculation in memory, the theoretical sequential runtime would require 37.3 GB of memory on a single system. With this in mind, the BOINC implementation not only offers the reduced runtime, but also the aggregation of the available memory of all participant nodes. If an actual sequential run of the problem were compared, a more drastic reduction in the runtime would be seen due to an additional secondary storage I/O overhead for a practical system. Despite the limitations of the public computer architecture, most notably in communication overhead, it represents a practical platform for grid- and cluster-scale bioinformatics computations today and shows great potential for future implementations.

Index Terms—Basic local alignment search tool (BLAST), Berkeley Open Infrastructure for network computing (BOINC), gene sequence alignment, public computer.

I. INTRODUCTION

BIOINFORMATICS offers challenging problems that can benefit from the vast resources of highly parallel distributed systems such as public computers. Foundational problems such as global gene sequence alignment exhibit substantial computation and memory requirements to both calculate and store solutions [1]. More practical tools such as the popular Basic Local Alignment Search Tool (BLAST) [2] encounter ever increasing database sizes as well as growing queries both in individual query size and number of queries. Public computing is

Manuscript received February 26, 2007; revised October 25, 2007. The work of S. Pellicer was supported by the Molecular Basis of Disease Fellowship Grant. The work of G. Chen was supported in part by the National Science Foundation (NSF) of China under Grant 60573131, Grant 60673154, and Grant 60721002, in part by Jiangsu Provincial NSF under Grant BK2005208 and Grant BG2007039, and in part by the China 973 projects (2006CB303000). The work of Y. Pan was supported in part by the U.S. NSF under Grant CCF-0514750 and Grant CCF-0646102. *Asterisk indicates corresponding author.*

*S. Pellicer is with the Department of Computer Science, Georgia State University, Atlanta, GA 30302-3965 USA (e-mail: spellicer1@student.gsu.edu).

Y. Pan is with the Department of Computer Science, Georgia State University, Atlanta, GA 30302-3965 USA (e-mail: pan@cs.gsu.edu).

G. Chen is with the State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210093, China (e-mail: gchen@nju.edu.cn).

K. C. C. Chan is with the Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong (e-mail: cskchan@comp.polyu.edu.hk).

Digital Object Identifier 10.1109/TNB.2008.2000148

emerging as a viable architecture for tackling large-scale computational problems. Work with the public computer architecture has focused on computations with little task interdependency, light communication requirements, and problems with virtually limitless work. Our research examines the viability of the public computer architecture for bioinformatics applications on a scale normally targeted at shared memory multiprocessor machines and computational clusters. A theoretical sequential run of the global gene sequence alignment of two 100 000 base sequences requiring about 37.3 GB of memory for an in-memory computation would take 1.15 times our public computer implementation on 30 heterogeneous nodes. The sequential version of the BLAST would take 16.76 times our public computer implementation on 26 heterogeneous nodes. Given the limitations of the public computing paradigm, the platform shows promise as an alternative to traditional parallel computing architectures for common tasks in the field of bioinformatics.

Public computing, or volunteer computing, is an architecture designed to harness the idle cycles of Internet-connected workstations. The Berkeley open infrastructure for network computing (BOINC) is a general framework designed to implement this new architectural paradigm. The system software is being developed at the Space Sciences Laboratory at the University of California, Berkeley (UCB), with project leader Anderson [3]. This system is currently deployed for the search for extraterrestrial intelligence (SETI)@home project and a growing number of additional projects. The system is designed as a software platform utilizing computing resources from volunteer computers.

The global gene sequence alignment is a challenging task that serves as an initial step in many of the problems in bioinformatics. This problem is often addressed by using dynamic programming algorithms [1]. The use of dynamic programming involves large matrices containing the scoring for alignment with substantial computation and even more demanding memory requirements. In addition to these demands, the problem exhibits an interesting structure of task dependency constraining the amount of parallelization available in the computation. If scoring matrices are stored on the computational nodes, the communication overhead of the problem is relatively light. Our implementation of a distributed global gene sequence alignment application divides the scoring matrix into smaller submatrices. Each submatrix is solved, and the results required for adjacent submatrices are returned to a central server for coordinating the computation of these adjacent submatrices.

The popular BLAST is a bioinformatics data mining application that exhibits large computation costs and data parallelism well suited to the public computing architecture. BLAST is a tool used to find gene or protein sequences in a database

similar to a query sequence. This involves performing local alignments between the query and database sequences, and calculating the statistical significance of similarity [2]. While one can exploit the parallelism offered by a public computer for the computational tasks associated with BLAST, the large amounts of database data involved with a typical query represent a challenge for the weak communication infrastructure offered in current public computer architectures. Our implementation of a distributed BLAST application divides the target database into roughly equally sized compressed units. Participant nodes download and decompress a section of the database and a batch of queries, run the BLAST queries against the database section, compress the results, and return the results to the central server.

Our research uses BOINC as a platform for performing bioinformatics computations using distributed heterogeneous nodes. These distributed applications implemented in this paper differ from existing public computer applications in scale. Problems of the sizes found in this paper are normally solved using shared memory multiprocessor systems or computing clusters. The public computer architecture is normally used for problems of sizes intractable on traditional parallel and distributed architectures. Using the public computer architecture for problems of the size examined in this paper offers the potential for increased performance and/or reduced cost of high-performance computing in bioinformatics.

II. BACKGROUND AND RELATED WORKS

Current research in public computing concentrates on projects of sizes that are, for the most part, intractable on systems of the typical computing cluster size. On the other hand, the parallelization of bioinformatics experiments and applications focus on shared memory and cluster implementations. A number of large computing projects utilize the public computing architecture. This section describes the general architecture and a selection of current public computing projects. Additionally, this section describes the sequential approaches and previous parallelization approaches to the two bioinformatics problems of interest: the global gene sequence alignment and the BLAST data mining tool.

A. Public Computing

Public computing is an approach to distributed computation that coordinates the resources of heterogeneous, geographically distributed computers. Public computing is also referred to as Internet computing or volunteer computing. Early public computing projects were responsible for implementing both the public computer framework as well as the code necessary for processing the project work. BOINC was developed as a generic framework for implementing public computing applications. BOINC implements most of the software necessary for serving a public computing project while individual projects supply the project specific code. The system software is being developed at the Space Sciences Laboratory at the UCB, with project leader Anderson [4]. This system is currently deployed for the SETI@home project and a growing number of additional

projects. The system is designed as a software platform utilizing computing resources from volunteer computers.

Outside of public computing, distributed computation has seen the emergence of many architectures including grid and computing clusters. Public computing differs from the other approaches in two major areas: the use of resources that may cross organizational administrative domains with little or no accountability and architectural differences including communication constraints, heterogeneity, and reliability. In an interview with the Association for Computing Machinery (ACM) Queue, Anderson, the director of the SETI@home and BOINC projects at UCB Space Sciences Laboratory, offers a very concise explanation of the differences in accountability [5]:

It's particularly important to distinguish between grid computing and volunteer computing. Grid computing involves the sharing of resources within or between organizations such as universities, research labs, and companies. The resources are secure, trusted, and centrally managed. The organizations are accountable to each other, and their relationships are symmetric.

Volunteer computing, on the other hand, involves an asymmetric relationship: participants donate their CPU time to projects. Participants aren't accountable to projects, so the projects can't trust them. The infrastructure software must reflect this, as evidenced by the redundancy mechanism I mentioned earlier.

This asymmetric relationship between volunteer computational resources and the project introduces unique challenges to the public computer architecture. Because the computational nodes are outside the project administrative control, work done by the resources must be verified as correct. A common technique for verifying work is redundantly computing the same work unit on different nodes. These redundant results are compared in order to find errors or purposefully incorrect results. Taufer *et al.* discusses a greater challenge encountered by their Predictor@Home protein structure prediction application where comparison of redundant results is dependent on the floating point operation of the heterogeneous platforms [6]. Different CPU architectures perform floating point computation with varying amounts of accuracy. As such, Predictor@Home must coordinate performing redundant computation on homogenous architectures in order to successfully compare results.

Due to the large scale of many of the current public computer applications, problems such as task distribution speed and tracking long running tasks are of concern. Chistensen *et al.* encountered the problem of reporting task progress on tasks for their climateprediction.net project that can typically run for 840 h on a 1.6 GHz Pentium 4 [7]. Their work examines the overhead of application *checkpointing* to save the progress of long running tasks in order to continue tasks after interruption. Additionally, their project utilizes a "trickle" mechanism to report incremental progress on a task back to the project. This gives users a more immediate feedback of progress for long running tasks. Anderson *et al.* study the performance of serving tasks in large projects such as SETI@home for distributing a load of tasks on the order of 8.8 million tasks per day on an even modest server hardware [8].

Current research in public computing systems are similar to these cited projects. These projects typically deal with problems

	A	A	C	G	T	T	A	G
A		1	0	0	0	0	1	0
A	1		0	0	0	0	0	0
G	0	0			0	0	0	1
T	0	0	0	0		1	0	0
T	0	0	0	0	1		1	0
A	1	1	0	0	0	1		2
G	0	0	0	1	0	0	2	
G	0	0	0	1	0	0	0	3

Fig. 1. Complete scoring matrix.

of a large scale having workloads of a virtually limitless size. These projects deal with issues resulting from thousands of participants with work units that can last weeks or even months. This work utilizes the middleware of these types of projects while targeting problems of a more cluster-sized scale.

B. Global Gene Sequence Alignment

The global gene sequence alignment is a fundamental task in working with genome information. The goal of this calculation is to align gene sequences according to a scoring system by penalizing gaps and mismatches between the sequences. Sequences can be aligned in a number of ways taking into account inserting gaps and accounting for mismatched bases. A scoring system can penalize the addition of gaps and mismatches. Different alignments will produce different scores.

The sequential algorithm uses a dynamic programming approach to calculate the score for all possible alignments simultaneously [1]. The scale of this problem for two sequences of length n and m is $O(nm)$ for both the number of comparisons required and the memory used. The algorithm produces an alignment matrix as its final output with the scores for each alignment in each cell of the matrix. The alignment matrix takes each base of the first gene sequence as a column heading and each base of the second gene sequence as a row heading (Fig. 1).

The algorithm then begins at the upper left corner of the matrix. The base from the column heading is compared to the base of the row heading. A match is given a score (1) while a mismatch is given a penalty (-1). This match adjustment m is combined with the score from the upper left neighbor. Additionally, the scores from the upper neighbor and left neighbor are each combined with a gap penalty (-2). The score $n_{i,j}$ for row i and column j is calculated with the formula

$$n_{i,j} = \max(0, (n_{i-1,j-1} + m), (n_{i-1,j} - 2), (n_{i,j-1} - 2)). \quad (1)$$

The resulting scores express the fitness of a particular alignment based on the gap and matching penalties set in the calculation.

Once all the scores are calculated, the matrix can be searched for the highest score and trace back through the neighboring values to find the best sequence alignment. The dotted line in Fig. 1 represents the best alignment of the two sequences.

Our initial experiments with implementing this dynamic programming algorithm on the public computing platform concen-

trated on comparing the performance of scheduling algorithms given the highly interdependent nature of the calculation [9]. The parallel performance of these types of problems suffers greatly compared with similar sized computations featuring highly independent tasks [10], [11].

C. BLAST

The BLAST is a method of ascertaining gene or protein sequence similarity [2]. The program takes a query sequence and searches against a database selected by the user. It aligns a query sequence against every sequence in the database. The results are reported in the form of a ranked list followed by a series of individual sequence alignments, plus various statistics and scores. Every hit in that list is assigned a similarity score S . Further, the score is analyzed for how likely it is to arise by chance. For that purpose, the so-called E -value is calculated for every hit. The E -value for a score S represents the expected number of hits of the score S or higher in the database.

There have been a number of efforts to parallelize BLAST for increased performance.

mpiBLAST uses the message passing interface (MPI) to parallelize BLAST by querying segments of the target database simultaneously on a computing cluster [12]. This implementation preprocesses the target database by dividing it into unique segments for use by individual nodes in the cluster. By preprocessing the database into segments, mpiBLAST can copy the segment to the cluster node, reducing communication with shared storage during the computation. In tests on up to 128 nodes of a 240 node Linux Beowulf cluster, mpiBLAST achieved superlinear speedup versus a single worker node. These tests pre-distributed database segments to nodes and ran short preliminary queries on each node to prime caches on the system. These conditions are meant to simulate a cluster processing many BLAST queries in succession. While mpiBLAST offers superlinear speedup in these tested conditions, efficiency decreased as the number of nodes increased.

Lin *et al.* [13] investigate improving mpiBLAST through an optimization of the I/O and communication. Their implementation of a parallel BLAST, pioBLAST, features online, dynamic partitioning of the database, parallel I/O through the use of an MPI-IO, and efficient merging of results. This implementation addresses many of the scalability issues of mpiBLAST. As mpiBLAST encounters more processing nodes or changes to the target database, the issue of preprocessing the database into segments and the granularity of the problem becomes a performance issue. The super-linear speedup of mpiBLAST depends heavily on the database being segmented outside the observed runtime. pioBLAST dynamically partitions the database into fragments that can effectively run from memory on cluster nodes eliminating the on-disk copies of intermediate segments. The results of pioBLAST benchmarks drastically reduces the amount of I/O overhead compared to mpiBLAST when accounting for the step of database segmentation.

The contrast between these two cluster implementations points out two issues with data parallel BLAST implementations. First, the dynamic partitioning of the database in

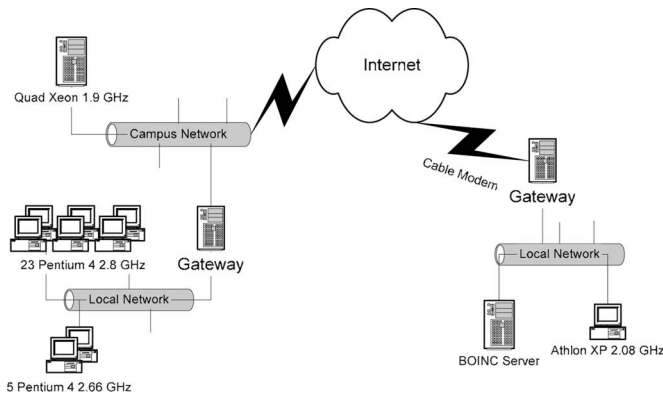


Fig. 2. Network diagram.

pioBLAST exhibits the desirable attribute of a centralized data store. This would allow for central updates to the database as data quality increases through the correction of data and discovery of new knowledge. A major increase of performance for mpiBLAST is the repartitioning and local storage of the database. Second, pioBLAST gains additional performance during sorting of results by overlapping the writing of sets of results as computational nodes report back to the master node. These improvements come from both a high-performance parallel I/O library and the rich communication available on the computing cluster.

III. BOINC IMPLEMENTATION

Our implementation of two bioinformatics applications on the BOINC platform harnesses the resources of a heterogeneous set of personal computers using a modest project server. The gene sequence alignment application and BLAST application in this paper execute on slightly different sets of computing nodes due to the availability of systems and software at the time of the experiments.

The general architecture of the computers and network used in both experiments are shown in Fig. 2. The largest group of workstations consists of 28 Linux workstations. Most of these workstations have 2.8 GHz Intel Pentium 4 processors except five systems with Intel Pentium 4 2.66 GHz processors. These workstations are on a shared local network connected to the larger campus network via a gateway firewall. A Quad Intel Xeon 1.9 GHz system residing on the campus network was also used. The BOINC project server resides on a network outside the campus network connected to the Internet via a cable modem. An additional Athlon XP 2.08 GHz running Microsoft Windows XP resides on the remote network where the BOINC project server resides.

Benchmarks of the network bandwidth between different systems was measured by timing transfers of large files. The Quad Intel Xeon system used in the BLAST application for serving data files offers roughly 1.17 MB of upload bandwidth to the large group of workstations. The connection from the campus network to the BOINC server on a different network offers 1009.54 KB of bandwidth. This system configuration offers communication performance somewhere between the two popular extremes of distributed computing: high-performance cluster

environments and highly distributed, large-scale public computer environments. Cluster environments often feature interconnection networks offering local area network speeds of 100 MB/s or more, while current public computer implementations are normally for loosely coupled nodes typically scattered across networks of varying performance over a wide geographic area.

Scheduling for both applications was done on a first come, first serve (FCFS) fashion. Each compute node was allowed to process one workunit at a time. Requests for more workunits were denied until outstanding workunits were completed, results uploaded, and results recorded. Due to limitations with the default BOINC client for computations of a smaller scale than intended, compute nodes do not necessarily report results immediately upon the workunit completion. This design is suited for the massive scale computations for which the BOINC architecture was designed, as it reduces the amount of contact with the server by the compute nodes. In smaller, cluster-scale computations such as this experiment, this design feature leaves compute nodes occasionally idle during the period between completing a workunit and reporting results to the server. To reduce the impact of this feature, compute nodes were occasionally polled to report results to the server. The requirement for this workaround likely lowered the performance of the application, especially when larger numbers of workunits were used. A feature for lower latency projects with shorter reporting times is being implemented in the BOINC framework, but was not available for either of the experiments in this paper.

A. BOINC Gene Sequence Alignment

The parallel BOINC implementation of the sequence alignment algorithm divides the solution matrix into equally sized submatrices and calculates the solutions on the participant nodes. To handle the dependencies of adjacent submatrices, the server complex relays the adjacent columns and rows to new compute nodes as they are completed. The final solution matrices are not transferred back to the central server. The experiment implemented for this study does not search the solution matrix for the best alignment. The focus of the experiment is on the task dependency of generating the solution matrix. While returning to the solution matrix to find the final alignment is an important aspect of the calculation, the version of the BOINC software used in the experiment lacked a feature under development that would allow leaving the solution matrix on the client nodes. The missing feature of persistent file storage on client nodes would allow the BOINC implementation to leave the solution matrix on the compute node and send later tasks for alignment retrieval back to nodes containing the appropriate solution segment. While searching these solution matrices is not explored, the solution matrix is written to disk on the client in order to account for disk write times in the benchmarks.

Based on the score formula, the score for an entry in the alignment matrix depends on the scores calculated for its previously calculated neighbors. Fig. 3 illustrates the dependencies of this calculation.

The squares in the diagram represent the cells of the alignment matrix, whereas the solid arrows represent the dependencies of

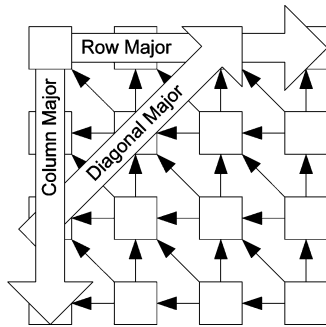


Fig. 3. Sequence alignment task order.

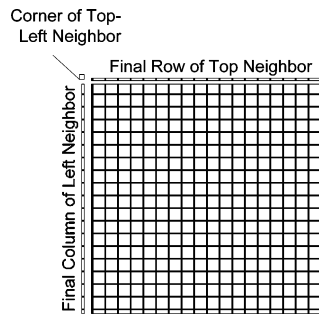


Fig. 4. Workunit input.

the cell with its neighbors. The top and left border cells assume a zero score for the nonexistent neighbors. The algorithm can progress through the calculation of cells either in row-major, column-major, or diagonal-major order to satisfy the dependencies of each calculation (the hollow arrows shown in Fig. 3).

Each workunit represents a submatrix of the final solution matrix. The final solution matrix of dimensions $100\,000 \times 100\,000$ is divided into 2500 equal parts using submatrices of dimensions 2000×2000 . The input file contains the position of the submatrix in the final solution matrix and the dimensions of the submatrix.

Three additional files are included in each workunit containing the values of the column left-adjacent to the workunit, the values of the row top-adjacent to the workunit, and the value of the corner top-left-adjacent of the workunit (Fig. 4).

The gene sequences are not included in each workunit download. Instead, both sequences are transferred in their entirety with the client program. This reduces the redundant data transferred to the client as each workunit will reuse these data throughout the application.

As stated previously, the full alignment matrix is not included with the results sent back to the central server. The results are written to a disk on the client node, and are required for the completion of a workunit. To reduce communication costs, the result unit sends back only the right column, bottom-right corner cell, and bottom row to the server (Fig. 5). These three outputs are necessary for the server to generate the new workunits dependent on the current workunit.

The BOINC server of the sequence alignment experiment generates new workunits as the required dependencies are satisfied. Upon receiving the results of a completed workunit, the

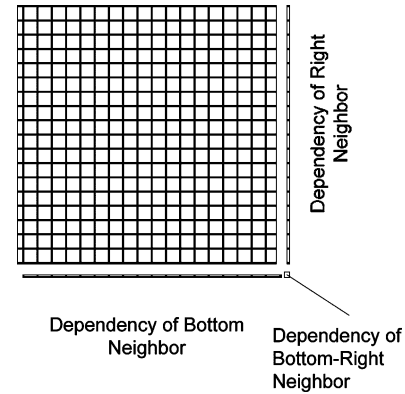


Fig. 5. Result unit output.

server checks its records for required inputs of neighboring workunits. If a workunit is found with all three required inputs (top-adjacent row, left-adjacent column, and top-left adjacent corner), the server generates the new workunit input file and transfers the required inputs to the download directory of the server. Compute nodes contacting the BOINC server can then download these input files and the new workunits.

B. BOINC BLAST

This problem was chosen because it exhibits an attribute with a tremendous negative impact to public computing: heavy communication overhead. The problem offers very straightforward data parallelism through the division of the target database; however, this database is large and must be transferred to the target computational nodes. In addition, the matching results and statistical data from a query on a portion of the database must be accumulated and returned to the BOINC server complex. This experiment concentrates its analysis on the communication issues involved in the application and the reduction in the job execution time. The communication issues include the data transfer bandwidth available to the public computer and the overlapping communication and computation.

Implementing BLAST on the BOINC platform exploits data parallelism on the target database and a batch of queries against these database fragments. Each workunit of the public computer application executes an entire batch of queries against a single fragment of the database and returns the results to the central server complex (Fig. 6). This application design has two areas of high communication: downloading of the database fragment to a compute node from the database server and uploading the query results to the central server complex. Each workunit performs its batch of queries using code similar to the sequential BLAST tool using default query and output parameters with modifications for the BOINC application program interface (API) calls for managing input and output files and initializing the API.

For each workunit, the client downloads a compressed segment of the database. The segment is decompressed and the entire group of queries is executed against the segment. The text output of the query is then compressed and uploaded to the BOINC server.

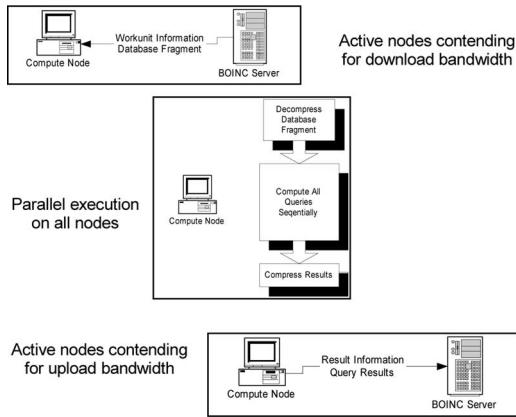


Fig. 6. Communication and computation.

It should be noted that this implementation currently does not address one of the issues addressed in the previous cluster-based work such as *pioBLAST*: merging and sorting of final results. This issue would benefit from an approach similar to that found in *pioBLAST*. Final results would need to be merged into a sorted list as they were reported from computation nodes. While the analysis of this implementation does account for the communication overhead of reporting results from computational nodes, the final merge of the results is not implemented and would incur an additional final computational cost on the central server. However, because all database segments are uploaded to computation nodes, this implementation would still allow for central updates to the database as in *pioBLAST* and unlike *mpiBLAST*. While this implementation prepartitions the database, it is still maintained on a central file distribution point. It is not unfeasible to partition the database as workunits are requested. This feature would allow for regular updates to the database with little impact to performance.

IV. RESULTS

A. BOINC Gene Sequence Alignment

The task dependency of the calculation produces interesting speedup results due to the amount of parallelism available in the problem initially growing to a peak, and then, dwindling back to sequential.

The compute nodes used in this experiment are 23 of the 2.80 GHz Linux Workstations, 5 of the 2.66 GHz Linux Workstations, the Quad XEON Workstation, and the Athlon Workstation. The BOINC server is used for serving workunits, data files, and receiving result unit output files.

The sequential performance measures used in this experiment are extrapolated from the calculations needed for a subset of the entire problem. This experiment aligns two generated sequences each of length 100 000. Because the theoretical sequential program would benefit from performing the entire calculation in memory, extrapolated results are acceptable because this theoretical maximum performance would require a machine with 37.3 GB of memory in order to store the entire solution matrix if each cell required a 32 B integer. Even with memory reduction techniques for storing the sparse solution matrix, the memory

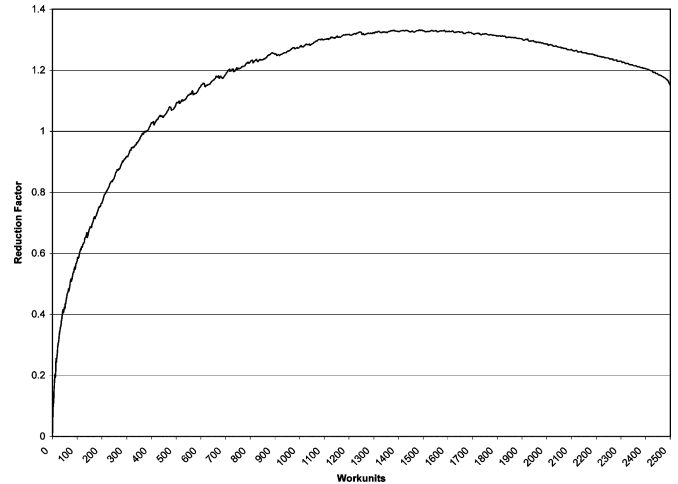


Fig. 7. Sequence alignment runtime reduction factor.

required would be many gigabytes. The extrapolated sequential benchmark times only account for the generation of the solution matrix and do not include searching the matrix for the final sequence alignment.

Examination of the results must be viewed in the context of the extrapolated sequential running times. These times are based on the best CPU time of a single-workunit-sized alignment. This time is extrapolated to 2500 workunits for the total execution of the problem sequentially for a total runtime of 5000 s. This extrapolated time would require an in-memory computation of the entire problem in 37.3 GB of memory. A realistic sequential execution would turn to secondary storage increasing the run time. The parallel runtime represents all computation and communication costs associated with an actual execution.

Tasks must be completed in diagonal-major order in order to satisfy the dependencies for later calculations. At step one in the calculation, only one workunit is available. Step two offers two workunits; step three offers three; and so on. This trend continues until the largest diagonal of the matrix is reached, and then, the number of tasks that can be completed in parallel then starts to increase downward until the lower right corner of the matrix is reached. The parallel implementation can process each of these diagonals in parallel.

Fig. 7 shows the ratio of sequential runtime to parallel runtime.

The reduction factor graph shows the amount of the runtime reduction based on the number of workunits completed. The sequential runtime increases linearly as the amount of work increases; however, the parallel algorithm runtime varies based on the amount of parallelization offered by the problem at various stages. The parallel algorithm is slower than the sequential algorithm on a single processor until about 400 workunits are completed. This slowdown is due to the overhead of the public computer implementation such as communication costs, scheduling, computation startup, and teardown, as well as reporting delays by the client software. Toward the middle of the problem, we achieve the greatest reduction factor in runtime due to the maximum parallelization of the problem while computing the major diagonal of the matrix of submatrices. After

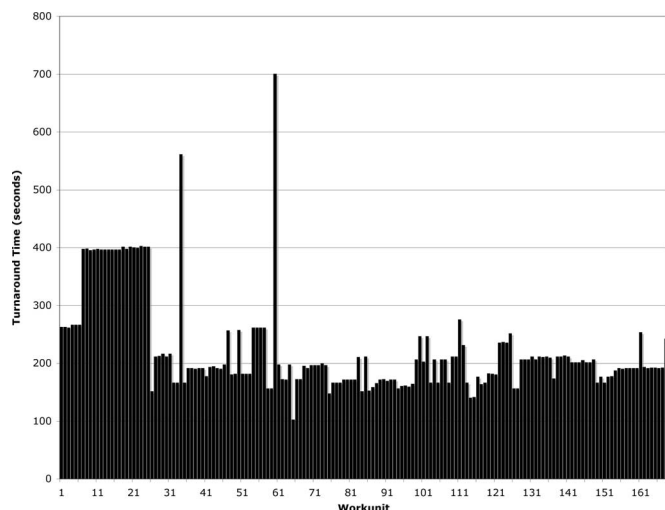


Fig. 10. BLAST workunit turnaround time.

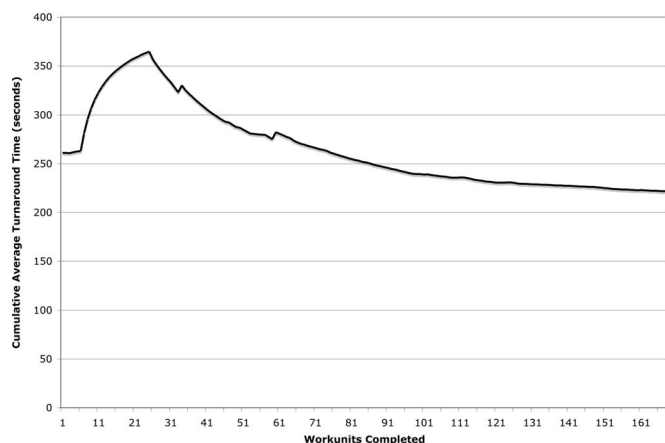


Fig. 11. BLAST workunit cumulative average turnaround time.

central server storing the final results. Fig. 10 shows the actual workunit completion time sorted by the arrival time at the central server. The earliest workunits show turnaround times significantly higher than the estimate. As the job progresses, the workunit turnaround time varies with many well below the estimate. This is likely due to an overlap in the computation and the communication by the varied start and finish times of the compute nodes as well as varying computation times for specific sections of the database. Many nodes will be computing and not contending for bandwidth as results are uploaded and new database segments are downloaded. Fig. 11 shows the cumulative average workunit turnaround time for the number of workunits completed. The job begins with shorter turnaround times as nodes join the computation. The average hits its peak as the first batch of 26 workunits complete with all participating nodes in relative synchronization. As the job progresses, the cumulative average continues to drop completing at an average well below the conservative estimates.

V. DISCUSSION AND FUTURE WORK

The implementation of these two bioinformatics applications on the public computing architecture offer distinct benefits. The BOINC global gene sequence alignment application offers reduced runtime as well as the benefit of aggregating the memory available of participating nodes. The BOINC BLAST application offers significant reduction in runtime using loosely coupled, uncoordinated systems. Both applications face significant hurdles for the public computing architecture: gene sequence alignment exhibits highly interdependent workunits and BLAST requires significant communication overhead.

Our previous work in utilizing BOINC for problems of similar scale to these two bioinformatics applications illustrates the impact of the two factors of task interdependency and communication overhead. The results of the BOINC BLAST experiment are consistent with our previous work, with an application that calculates the avalanche photodiode gain (APD) and the impulse response with a similar profile: heavy computation coupled with high communication cost [11]. BOINC BLAST exhibited greater speedup than our APD application most likely due to moving data servers to networks with greater bandwidth to the participating nodes. Our results are also consistent with our previous cryptographic application that featured task independence and negligible communication overhead [10]. This cryptographic application exhibited drastic speedup compared to these two bioinformatics applications illustrating the impact of task interdependence and significant communication overhead.

Future work for experiments such as this should concentrate on enriching the communication capabilities of public computing and scaling down the scheduling and reporting mechanisms of the platform. Peer-to-peer communication capabilities could reduce the impact of multiple nodes contending for bandwidth with data and scheduling servers. In the case of global gene sequence alignment, peer-to-peer capabilities would enable an implementation that allows clients to contact peer clients for task dependency (i.e., final column, row, and corner output of adjacent tasks). For BLAST, peers could share database information with each other in order to reduce the contention for bandwidth with a central data server. Client software with more rapid reporting mechanisms could increase performance in smaller scale computations such as this experiment by reducing the idle time of compute nodes after the workunit completion. Improvements to scheduling for computations of this scale could reduce the impact of scheduling overhead.

VI. CONCLUSION

The public computer architecture shows promise as a platform for solving fundamental problems in bioinformatics such as global gene sequence alignment and data mining with tools such as BLAST. Our implementation of these two problems on the BOINC platform demonstrates a runtime reduction factor of 1.15 for sequence alignment and 16.76 for BLAST. While the runtime reduction factor of the global gene sequence alignment application is modest, this value is based on a theoretical sequential runtime extrapolated from the calculation of a smaller

problem. Because this runtime is extrapolated from running the calculation in memory, the theoretical sequential runtime would require 37.3 GB of memory on a single system. With this in mind, the BOINC implementation not only offers the reduced runtime, but also the aggregation of the available memory of all participant nodes. If an actual sequential run of the problem was compared, a more drastic reduction in runtime would be seen due to an additional secondary storage I/O overhead for a practical system. Despite the limitations of the public computer architecture, most notably in the communication overhead, it represents a practical platform for grid- and cluster-scale bioinformatics computations today, and shows great potential for future implementations.

REFERENCES

- [1] D. Sankoff, "The early introduction of dynamic programming into computational biology," *Bioinformatics*, vol. 16, no. 1, pp. 41–47, 2000.
- [2] S. Altschul, W. Gisha, W. Millerb, E. Meyersc, and D. Lipmana, "Basic local alignment search tool," *J. Mol. Biol.*, vol. 215, no. 3, pp. 403–410, Oct. 5, 1990.
- [3] D. Anderson, "Public computing: Reconnecting people to science," presented at the Shared Knowl. Web Conf., Madrid, Spain, Nov. 2003.
- [4] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proc. 5th IEEE/ACM Int. Workshop Grid Comput.*, 2004, Nov., pp. 4–10.
- [5] J. Maurer, "A conversation with david anderson," *Queue*, vol. 3, no. 5, pp. 18–25, Jul. 2005.
- [6] M. Tauffer, C. An, A. Kerstens, and C. L. Brooks, III, "Predictor@home: A protein structure prediction supercomputer based on public-resource computing," presented at the 19th IEEE Int. Parallel Distrib. Process. Symp., Apr. 2005, Denver, CO.
- [7] C. Christensen, T. Aina, and D. Stainforth, "The challenge of volunteer computing with length climate model simulations," in *Proc. 1st Int. Conf. e-Sci. Grid Comput.*, 2005, Dec., pp. 8–15.
- [8] D. P. Anderson, E. Korpela, and R. Walton, "High-performance task distribution for volunteer computing," in *Proc. 1st Int. Conf. e-Sci. Grid Comput.*, 2005, Dec., pp. 196–203.
- [9] S. Pellicer, N. Ahmed, Y. Pan, and Y. Zheng, "Gene sequence alignment on a public computing platform," in *Proc. ICPP 2005*, Jun., pp. 95–102.
- [10] S. Pellicer, Y. Pan, and M. Guo, "Distributed MD4 password hashing with grid computing package BOINC," presented at the 2004 Int. Conf. Grid Cooperative Comput. (GCC-04), Oct., Wuhan, China.
- [11] S. Pellicer, Y. Pan, P. Sun, and M. Hayat, "Avalanche photodiode gain and impulse response calculation on a public computing platform," in *Proc. 19th IEEE Int. Parallel Distrib. Process. Symp.*, 2005, p. 256a.
- [12] A. E. Darling, L. Carey, and W. Feng, "The design, implementation, and evaluation of mpiBLAST," presented at the ClusterWorld Conf. Expo 4th Int. Conf. Linux Clusters: HPC Revolution 2003, San Jose, CA.
- [13] H. Lin, X. Ma, P. Chandramohan, A. Geist, and N. Samatova, "Efficient data access for parallel BLAST," presented at the 19th IEEE Int. Parallel Distrib. Process. Symp., Apr. 2005, Denver, CO.



Guihai Chen (M'05) received the B.S. degree in computer science from Nanjing University, Nanjing, China, in 1984, the M.Eng. degree in computer engineering from Southeast University, Nanjing, in 1987, and the Ph.D. degree in computer science from the University of Hong Kong, Kuwloon, Hong Kong, in 1997.

During 1998, he was a Research Fellow on a visiting assignment at the Kyushu Institute of Technology, Kitakyushu, Japan. During 2000, he was a Visiting Professor at the University of Queensland, Qld., Australia. During September 2001 to August 2003, he was a Visiting Professor at Wayne State University. He is currently a Full Professor and the Deputy Chair in the Department of Computer Science, Nanjing University. He is the author or coauthor of more than 100 papers published in peer-reviewed journals and refereed conference proceedings in the areas of wireless sensor networks, high-performance computer architecture, peer-to-peer computing and performance evaluation.

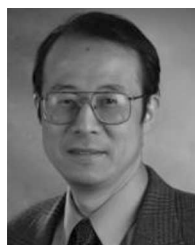
Prof. Chen is a member of the IEEE Computer Society. He has also served on technical program committees of numerous international conferences.



Keith C. C. Chan received the B.Math. degree in computer science and statistics, and the M.A.Sc. and Ph.D. degrees in systems design engineering from the University of Waterloo, Waterloo, ON, Canada, in 1984, 1985, and 1989, respectively.

He was with the IBM Canada Laboratory, Toronto, ON, where he was involved in the development of software engineering tools. In 1993, he joined the Department of Electrical and Computer Engineering, Ryerson University, Toronto, as an Associate Professor. In 1994, he joined the Department of

Computing, the Hong Kong Polytechnic University, Kuwloon, Hong Kong, where he is currently a Professor and the Head. He is also a Guest Professor at the Graduate University of the Chinese Academy of Sciences, Beijing, China. He is active in consultancy and has served as a consultant to government agencies and various companies in Hong Kong, China, Singapore, Malaysia, and Canada. His current research interests include data mining, bioinformatics, and software engineering.



Yi Pan (S'90–M'91–SM'91) received the B.Eng. and M.Eng. degrees in computer engineering from Tsinghua University, Beijing, China, in 1982 and 1984, respectively, and the Ph.D. degree in computer science from the University of Pittsburgh, Pittsburgh, PA, in 1991.

He is currently the Chair and a Professor in the Department of Computer Science and a Professor in the Department of Computer Information Systems, Georgia State University, Atlanta. His current research interests include parallel and distributed computing, networking, and bioinformatics.

He is the author or coauthor of more than 100 journal papers with over 30 papers published in various IEEE journals and over 100 papers in refereed conferences, and has coedited 33 books (including proceedings).

Dr. Pan was the IEEE Distinguished Speaker during 2000–2002, a Yamacraw Distinguished Speaker in 2002, and a Shell Oil Colloquium Speaker in 2002. He was the recipient of many awards from agencies such as the National Science Foundation (NSF), the Air Force Office of Scientific Research (AFOSR), the Japan Society for the Promotion of Science (JSPS), the International Information Science Foundation (IISF), and Mellon Foundation. He was an Editor-in-Chief or an editorial board member for 15 journals including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON NANOBIOSCIENCE, and the IEEE TRANSACTIONS ON SYSTEMS, MEN, AND CYBERNETICS. He was also the Guest Editor for ten journals including the IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS and IEEE TRANSACTIONS ON NANOBIOSCIENCE. He has organized many international conferences and workshops and has delivered over ten keynote speeches at international conferences. He is listed in *Men of Achievement*, *Who's Who in Midwest*, *Who's Who in America*, *Who's Who in American Education*, *Who's Who in Computational Science and Engineering*, and *Who's Who of Asian Americans*.



Stephen Pellicer received the B.S. degree in computer science in 1996 from the University of Georgia, Atlanta, and the M.S. degree in computer science in 2004 from Georgia State University, Atlanta, where he is currently working toward the Ph.D. degree.

His current research interests include parallel and distributed computing, peer-to-peer, and public computing systems with applications in high-throughput bioinformatics.

Mr. Pellicer is a Fellow of the Molecular Basis of Disease, Georgia State University.