EVALUATING PERFORMANCE OPTIMIZATIONS OF LARGE-SCALE GENOMIC SEQUENCE SEARCH APPLICATIONS USING SST/MACRO

Tae-Hyuk Ahn[1], Damian Dechev[2,3], Heshan Lin[1], Helgi Adalsteinsson[3] and Curtis Janssen[3]
[1]*Department of Computer Science, Virginia Tech, VA 24061, Blacksburg, U.S.A.*
[2]*Department of Electrical Engineering and Computer Science, University of Central Florida, FL 32816, Orlando, U.S.A.*
[3]*Scalable Computing R&D Department, Sandia National Laboratories, CA 94551, Livermore, U.S.A.*

Keywords: Exascale architecture simulator, mpiBLAST, Performance and scalability modeling.

Abstract: The next decade will see a rapid evolution of HPC node architectures as power and cooling constraints are limiting increases in microprocessor clock speeds and constraining data movement. Future and current HPC applications will have to change and adapt as node architectures evolve. The application of advanced cycle accurate node architecture simulators will play a crucial role for the design and optimization of future data intensive applications. In this paper, we present our simulation-based framework for analyzing the scalability and performance of a number of critical optimizations of a massively parallel genomic search application, mpiBLAST, using an advanced macroscale simulator (SST/macro). In this paper we report the use of our framework for the evaluation of three potential improvements of mpiBLAST: enabling high-performance parallel output, an approach for caching database fragments in memory, and a methodology for pre-distributing database segments. In our experimental setup, we performed query sequence matching on the genome of the yellow fever mosquito, *Aedes aegypti*.

1 INTRODUCTION

The exponential growth of data intensive applications and the necessity for complex and massive data analysis have elevated modern large-scale parallel computing technology and demand. Future High-Performance Computing (HPC) systems will go through a rapid evolution of node architectures as power and cooling constraints are limiting increases in microprocessor clock speeds. Consequently computer architects are trying to increase significantly the on-chip parallelism to keep up with the demands for fast performance and high volume of data processing. Multiple cores on a chip is no longer cutting edge technology due to this hardware paradigm shift. In the new Top 500 supercomputer list published in March 2011, more than 99% of supercomputers are multicore processors (Top500, 2011). As hardware has evolved, software applications must adapt and gain the capability of effectively running multiple tasks simultaneously through parallel methods. It is of critical importance to provide an accurate estimate of an application's performance in a massively parallel system both for predicting the most effective design of a multi-core large-scale architecture as well as for optimizing and fine-tuning the software application to efficiently execute in such a highly concurrent environment.

A key element of the strategy as we move forward is the co-design of applications, architectures, and programming environments, to navigate the increasingly daunting constraint-space for feasible exascale system design. The complexity of designing large-scale computer systems has motivated the development and utilization of a large number of cycle-accurate hardware and system simulators (Janssen et al., 2010; Underwood et al., 2007; Sherwood et al., 2002). There is a pressing need to develop accurate code analysis and system simulation platforms to insert application developers directly into the design process for HPC systems in the exascale era. It is of significant importance to build the simulation platforms for accurate emulation of the hardware architectures of the next decade and their design con-

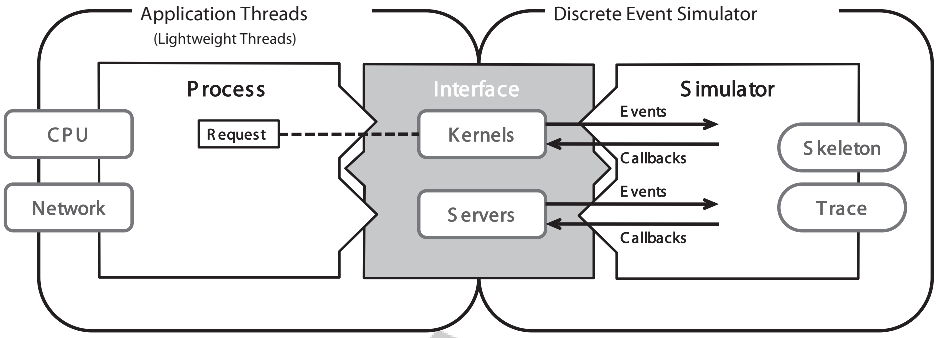

Figure 1: SST/macro Simulation Framework.

straints. This will enable computer scientists to engage early in the design and utilization of effective programming models.

Three well-known approaches have been investigated for estimating large-scale performance. The most common approach is direct execution of the full application on the target system (Prakash et al., 2000; Riesen, 2006; Zheng et al., 2005). This simulation approach uses virtual time unlike normal benchmarking that uses real time. Here, performance is modeled by using a processor model and communication work in addition to simulated time for a modeled network. Another approach is tracing the program in order to collect information about how it communicates and executes (Zheng et al., 2005). The resulting trace file contains computation time and actual network traffic. Tracing provides high levels of evaluation accuracy, but cannot be easily scaled to a different number of processors. A third approach is to implement a model skeleton program that is a simple curtailed version of the full application but provides enough information to simulate realistic activity (Susukita et al., 2008; Adve et al., 2002). This approach has the advantage that the bulk of the computational complexity can be replaced by simple calls with statistical timing information. What makes this approach challenging is the necessity to develop a model skeleton program based on a complex scientific HPC application that often includes a large number of HPC computational methods and libraries, sophisticated communication and synchronization patterns, and architecture-specific optimizations. Moreover, it is difficult to analyze and predict the runtime statistics for domain-specific applications using heuristic algorithms. The skeleton application provides a powerful method for evaluating the scalability and efficiency over various architectures of moderate or extreme scales. For example by running skeleton applications, the Structural Simulation Toolkit's macroscale simulator (SST/macro) (Janssen et al., 2010; SST/Macro, 2011) has been able to model application performance at levels of parallelism that are not obtainable on any known existing HPC system.

In this work we present the design and application of a discrete event simulation-based framework for analyzing the scalability and performance of a number of optimizations of mpiBLAST. mpiBLAST (Darling et al., 2003) is an open-source parallel implementation of the National Center for Biotechnology Information's (NCBI) Basic Local Alignment Search Tool (BLAST) (Altschul et al., 1990). BLAST is the most widely used genomic sequence alignment algorithm. Though a heuristic method is employed to improve computational efficiency, computation time is debilitating because of the rapid growth of sequence data. A parallel version of BLAST, mpiBLAST, uses a database segment approach. The design of mpiBLAST has been revised a number of times to better address the challenges of distributed result processing (Lin et al., 2005), hierarchical architectures (Thorsen et al., 2007), include further dynamic load balancing optimizations, and I/O optimizations (Lin et al., 2008; Lin et al., 2011). Our simulation-based framework allows programmers to better address the challenges of executing genomic sequence alignment algorithms on many-core architectures and at the same time gain important insights regarding the effectiveness of the mentioned mpiBLAST optimization techniques. This allows both scientists, library developers, and hardware architecture designers to evaluate the scalability and performance of a data intensive application on a wide variety of multi-core architectures, ranging from a regular cluster machine to a future many-core petascale supercomputer. Our approach can help in several ways including:

- enhance the evolution of the software application by performing further architecture-specific optimizations to meet the challenges of the communication and synchronization bottlenecks of the new multiprocessor architectures,

- adapt the hardware set-up to better facilitate the computational and communication patterns of the application,
- evaluate the effectiveness and associated trade-offs of any future co-design evolution of the application software and the hardware platform.

In this paper, we present the application of SST/-macro, an event-driven cycle-accurate macroscale simulator, for estimating and predicting the performance of large-scale parallel bioinformatics applications based on mpiBLAST. SST/macro has been recently developed and released by the Scalable Computing R&D Department at Sandia National Labs and is a fully component-based open source project that is freely available to the research and academic community (SST/Macro, 2011). We demonstrate the use of SST/macro and its trace-driven simulation that is based on DUMPI (Janssen et al., 2010), a custom-built MPI tracing library developed as a part of the SST/macro simulator. We also present a methodology for constructing SST/macro skeleton programs based on mpiBLAST.

The rest of this work is organized as follows: Section 2 introduces the event-driven SST/macro simulator that is at the core of our simulation framework, Section 3 discusses the mpiBLAST algorithm for parallel genome sequence matching and the possible optimizations of mpiBLAST, Section 4 presents in details the methodology of collecting DUMPI trace files and our approach for implementing mpiBLAST-based skeleton models as well as our experimental set-up and results, and Section 5 concludes this paper.

2 EVENT-DRIVEN MACROSCALE SIMULATION

We begin with a discussion of the high-level design and functionality of the SST macroscale simulator that is at the core of our framework. The overall network topology and model are presented in brief. Then we discuss MPI modeling through skeleton applications and MPI trace files.

The purpose of a large number of simulation tools and strategies is to help design new hardware platforms and better applications in HPC computing. The macroscale version of the Structural Simulation Toolkit is an architectural simulator that permits coarse-grained study of data intensive parallel scientific applications. SST/macro has a modular structure implemented in C++ (Stroustrup, 2000), allowing flexible addition of new components and modifications. Figure 1 shows the highlight of the design of the SST/macro simulator. The simulator makes use of extremely lightweight application threads, allowing it to maintain simultaneous task counts ranging into the millions. Task threads create communication and compute kernels, then interact with the simulator's back-end by pushing kernels down to the interface layer. The interface layer generates simulation events and handles the scheduling of resulting events to the simulator back-end. The interface layer implements servers to manage the interaction with the network model in the context of the application. SST/-macro supports two execution modes: skeleton application execution and trace-driven simulation mode. The processor layer receives callbacks when the kernels are completed.

Recent growth of large-scale systems has made evaluation of communication loads across complex networks vital. SST/macro is capable of simulating and evaluating advanced network workload with diverse topology and routing. The simulator currently supports torus, fat-free, hypercube, Clos, and gamma topologies, all described further in (Dally and Towles, 2003). Moreover, the general framework of a network can be easily evaluated with network parameters such as bandwidth and latency, thus allowing the capture of actual trade-offs between fidelity and runtime of a system's network. The routing algorithms are static in SST/macro, i.e., messages between two processors always follow the same path regardless of network status. The modularity of the simulator makes defining new connections easy.

2.1 The MPI Model

The Message Passing Interface (MPI) is a message passing library interface specification for a distributed parallel memory system (MPI, 2009). MPI primarily allows message-passing communication from the address space of one process to that of another process. MPI is not a language, and all MPI operations are expressed as functions, subroutines, or methods, according to the appropriate language bindings, for C, C++, and FORTRAN. The main advantages of MPI are portability and usability. The standard includes two main privileges: point-to-point message passing and collective operations. A number of important MPI functions involve communication between two specific processes based on point-to-point operations. MPI specifies mechanisms for both blocking and non-blocking point-to-point communication mechanisms. A procedure is blocking if returning from the procedure indicates the user is allowed to reuse resources specified in the call and a procedure is nonblocking if the procedure may return before the operation completes. Collective operation is defined as communication that involves groups of processors to invoke the procedure. These include such operations as all-to-all (all processes contribute and receive the result), all-to-one (all processes contribute to the result and one process receives the result), and one-to-all (one process contributes to the result and all processes receive the result).

In the SST/macro simulator, lightweight application threads perform MPI operations. SST/macro implements a complete simulated MPI which skeleton applications can use to emulate node communication in a direct manner. SST/macro has been used to test the performance impact of proposed extensions to the MPI standard. A simple processor model is added to provide timings for processor workload and data movement within each node. SST includes a network layer that supports a large array of interconnects. The application trace and CPU model helps to determine when a computation operation completes and schedules a completion event. The SST/macro components provide a complete performance estimation environment for HPC platforms.

2.2 Trace File Simulation

In the trace file simulation approach, an application is executed and profiled in order to extract a wealth of information about its execution pattern such as the average instruction mix, memory access patterns, and communication mechanisms and bottlenecks. The network utilization on a per-link basis is also estimated. The generated trace file contains data such as the time spent in computation and the communication footprints between processors. SST/macro supports two trace file formats: Open Trace Format (OTF) (Knüpfer et al., 2006) and DUMPI (Janssen et al., 2010). OTF is a trace definition and representation format designed for use with large-scale parallel platforms. The authors in (Knüpfer et al., 2006) identify three main design goals of OTF: openness, flexibility, and performance. DUMPI is a custom trace format developed as a part of the SST/macro simulator. Both of the trace formats record execution information by linking the target application with a library that uses the PMPI (Mintchev and Getov, 1997) interface to intercept MPI calls. The DUMPI format is designed to record more detailed information compared to OTF, including the full signature of all MPI-1 and MPI-2 calls. In addtion, DUMPI trace files store information regarding the return values and the MPI requests, which allows error checking and MPI operation matching. DUMPI files also provide processor hardware performance counter information using the Performance Application Programming Interface (PAPI) (Janssen et al., 2010), which allows information such as cache misses and floating point operations to be logged.

2.3 Skeleton Application Simulation

The main advantage of trace file driven simulation is accuracy, especially if the planned runtime system is known in details. However, a main difficulty is the fact that it requires the execution of the actual application that could often be data intensive and of high computational complexity. Moreover, trace file simulation is not capable of predicting performance on future hardware platforms, as the generated trace files are specific to the execution environment.

Skeleton applications are simplified models of actual HPC programs with enough communication and computation information to simulate the application's behavior. One method of implementing a skeleton application is to replace portions of the code performing computations with system calls that instruct the simulator to account for the time implicitly. Since the performance models can be embedded in the skeleton application and real calculations are not performed, the simulator requires significantly less computational cost than simulating the entire system. Skeleton application simulation can also evaluate efficiency and scalability at extremely different scales, which provides a powerful option for performance prediction of non-existing super-scalar systems. Though driving the simulator with a skeleton application is a powerful approach for evaluating the application's scalability and efficiency, it requires extensive efforts for programmers to implement the skeleton models for a large-scale parallel program. The effort is justified by the difficulty of predicting computation time for complex applications such as mpiBLAST. mpiBLAST search time varies greatly across the same size database and query, because the computation time depends on the number of positive matches found in a query. Match location can also affect the execution time.

Figure 2 shows the implementation of an MPI ping-pong skeleton application in which pairwise ranks communicate with each other. As shown in Figure 2, skeleton application implementations for the SST/macro are very similar to the native MPI implementation with the exception of the syntax of the MPI calls. In addition to replacing the communication calls, we can replace computation parts with system calls such as **compute(...)**, which reduce simulation time dramatically.

```
void mpipingpong::run() {
  timestamp start = mpi()->init();
  mpicomm world = mpi()->comm_world();
  mpitype type = mpitype::mpi_double;
  int rank = world.rank().id;
  int size = world.size().id;
  // With an odd number of nodes,
  // rank (size-1) sits out
  if (!((size % 2)&&(rank+1 >= size))){
    // partner nodes 0<=>1, 2<=>3, etc.
    mpiid peer(rank ^ 1);
    mpiapi::const_mpistatus_t stat;
    for (int half_cycle = 0;
         half_cycle < 2 * num_iter;
         ++half_cycle) {
      if ((half_cycle + rank) & 1)
        mpi()->send(count, type, peer,
                    tag, world);
      else
        mpi()->recv(count, type, peer,
                    tag, world, stat);
    }
  }
  timestamp done = mpi()->finalize();
}
```

Figure 2: Core execution loop of the MPI ping-pong skeleton application.

3 mpiBLAST

This section lays out the core design and functionality of mpiBLAST. Furthermore, we discuss the I/O and computation scheduling optimization proposed to mpiBLAST.

3.1 The Fundamental Design of mpiBLAST

In bioinformatics, a sequence alignment is an essential mechanism for the discovery of evolutionary relationships between sequences. One of the most widely used alignment search algorithms is BLAST (Basic Local Alignment Search Tool) (Altschul et al., 1990; Altschul et al., 1997). The BLAST algorithm searches for similarities between a set of query sequences and large databases of protein or nucleotide sequences. The BLAST algorithm is a heuristic search method for finding locally optimal alignments or HSP (high scoring pair) with a score of at least the specified threshold. The algorithm seeks words of length W that score at least T when aligned with the query and scored with a substitution matrix. Words in the database that score T or greater are extended in both directions in an attempt to find a locally optimal un-gapped alignment or HSP (high scoring pair) with a score of at least E value lower than the specified threshold. HSPs that meet these criteria will be reported, provided they do not exceed the cutoff value specified for the number of descriptions and/or alignments to report.

Today, the number of stored genomic sequences is increasing dramatically, which demands higher parallelization of sequence alignment tools. Moreover, next-generation sequencing, a new generation of non-Sanger-based sequencing technologies, has presented new challenges and opportunities in data intensive computing (Schuster, 2007). Many parallel approaches for BLAST have been investigated (Braun et al., 2001; Bjornson et al., 2002; Mathog, 2003; Lin et al., 2005), and mpiBLAST is an open-source, widely used parallel implementation of the NCBI BLAST toolkit.

The original design of mpiBLAST follows a database segmentation approach with a master/-worker system. It works by initially dividing up the database into multiple fragments. This preprocessing step is called `mpiformatdb`. The master uses a greedy algorithm to assign and distribute pre-partitioned database chunks to worker processors. Each worker then concurrently performs a BLAST search on its assigned database fragment in parallel. The master server receives the results from each worker, merges them, and writes the output file. mpiBLAST achieves an effective speedup when the number of processors is small or moderate. However, mpiBLAST suffers from non-search overheads when the number of processors increases and the database size varies. Additionally, the centralized output processing design can greatly hamper the scalability of mpiBLAST.

3.2 Optimizations of mpiBLAST

Hierarchical Architecture. mpiBLAST expands the original master-worker design to hierarchical design, which organizes all processes into equal-sized partitions by a *supermaster* process. The supermaster process manages assigning tasks to different partitions and handling inter-partition load balancing. There is one *master* processor for each partition that is responsible for coordinating both computation and I/O scheduling with many *workers* in a partition. This hierarchical design has an advantage in massive-scale parallel machines as it distributes the workload well across multiple partitions.

Dynamic Load Balancing Design. It is difficult to estimate the execution time of BLAST because search time is extremely variable and thus unpredictable

(Gardner et al., 2006). Therefore a greedy scheduling algorithm for fine-grained task assignment to idle processes is necessary. To avoid load imbalance while reducing the scheduling overhead, mpiBLAST adopts a dynamic worker group management approach where the masters dynamically maintain a window of outstanding tasks. Whenever a worker finishes its tasks, it requests further assignments from its master. With query prefetching, the master requests the next query segment when the total number of outstanding tasks in the window falls under a certain threshold.

Parallel I/O Strategy. Massive data I/O can lead to performance bottlenecks especially for data driven applications such as mpiBLAST. To deal with this challenge, mpiBLAST pre-distributes database fragments to workers before the search begins. Workers cache database fragments in memory instead of local storage. This is recommended on diskless platforms where there is no local storage attached to each processor. By default, mpiBLAST uses the master process to collect and write results within a partition, which may not be suitable for massively parallel sequential search. Asynchronous parallel output writing techniques optimize concurrent noncontiguous output access without inducing synchronization overhead which result from traditional collective output techniques.

4 EXPERIMENTAL RESULTS

We chose to identify and use freely available datasets for executing our mpiBLAST-based simulation analysis. In our experimental set-up we run mpiBLAST on the genome of the yellow fever mosquito, *Aedes aegypti*, which has been investigated by biologists for spreading dengue and yellow fever viruses. The genome database can be downloaded freely from the source in (Vectorbase, 2010) and has a suitable size of 1.4GB for testing on both our local machine and the cluster system. We use 1MB sequences randomly sampled from the *Aedes aegypti* transcriptome dataset because such query sequences match well with the genome's characteristics.

In our experiments, we relied on DUMPI to facilitate more detailed tracing of MPI calls than was available from other trace programs. The results of a DUMPI profiling run consists of two file formats. One is an ASCII metafile for the entire run, and the other is a binary trace file for each node. The metafile is a simple key/value ASCII file that is intended to be human-readable and to facilitate grouping related trace files together. Each trace file consists of a 64-bit lead-in magic number and 8 data records. In order to trace an application with DUMPI, a collection of DUMPI libraries are linked to the application when it is executed in the system. Afterwards, several executables built on DUMPI repository are used to analyze the DUMPI trace files.

In our experimental setup we have traced and analyzed the mpiBLAST implementation described in Section 3. The current open-source version of mpiBLAST has several options for parallel input/ouput of data. We have simulated and tested three optimizations as described below:

- Optimization 1 (--use-parallel-write), enabling high-performance parallel output: by default, mpiBLAST uses the master process to collect and write results within a partition. This is the most portable output solution and should work on any file system. However, using the parallel-write solution is highly recommended on platforms with fast network interconnection and high-throughput shared file systems.
- Optimization 2 (--use-virtual-frags), enabling workers to cache database fragments in memory instead of on local storage: this is recommended on diskless platforms where there is no local storage attaching to each processor.
- Optimization 3 (--predistribute-db), pre-distributing database fragments to workers before the search begins: especially useful in reducing data input time when multiple database replicas need to be distributed to workers.

We have traced a large number of mpiBLAST experimental executions with the SST/macro simulator to validate the simulator and predict the application's performance on a large-scale parallel machine. We executed our SST/macro simulation of the mpiBLAST application on two different platforms: a multi-core Linux machine and a distributed memory cluster system. The local machine consisted of a 2.66GHz Intel Core(TM)2 Duo CPU and 2GB memory. The cluster system is composed of 113 nodes, where each node contains two 3.2GHz Intel Xeon CPU and 2GB memory.

4.1 Validation of SST/macro with mpiBLAST

SST/macro has been recently released (SST/Macro, 2011) and it has not yet been exposed to testing outside of its development environment at Sandia National Labs. For this reason, in this section we briefly mention our findings in our efforts to validate the accuracy of the SST/macro simulator. The simulator was validated with mpiBLAST results using de-

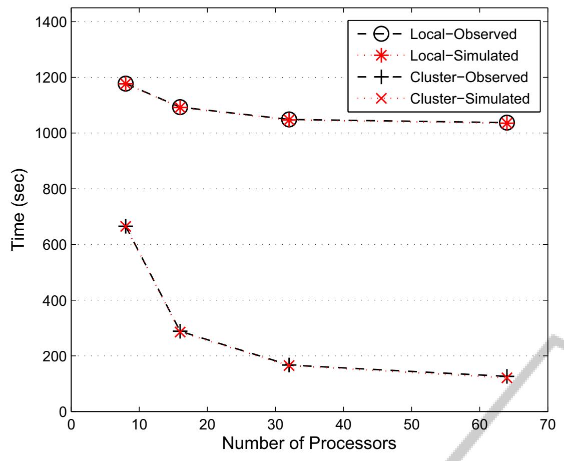

Figure 3: Comparison of observed and simulated runtimes both on the local machine and the cluster system.

fault bandwidth (2.5 GB/s) and latency (1.3 µs) on both testing environments. We used processor counts from 8 to 64, and traces were collected using the lightweight DUMPI library. Figure 3 shows the simulated walltime versus the elapsed realtime for the simulation driven by these DUMPI traces.

We applied the concept of K-L divergence (Emmert-Streib and Dehmer, 2008) to evaluate the similarity between the results from observed and simulated run time across the tested systems. K-L divergence is a non-commutative measure of the difference between two samples P and Q typically P representing the "true" distribution and Q representing arbitrary distribution. Therefore we set P as simulation results and Q as SST/macro DUMPI-driven runtimes with varying total CPUs. The K-L divergence is defined to be

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \qquad (1)$$

where $Q(i) \neq 0$. A smaller value of the K-L divergence variable signifies greater similarity between the two distributions.

Table 1 shows the K-L divergence distance. We carefully analyzed the resulting K-L distance and found out that the SST/macro trace clock times are very close to real simulation wall-times on both the local machine and the cluster system.

Table 1: The absolute distance and K-L divergence.

Method	**Local**	**Cluster**
K-L divergence	6.09	11.55

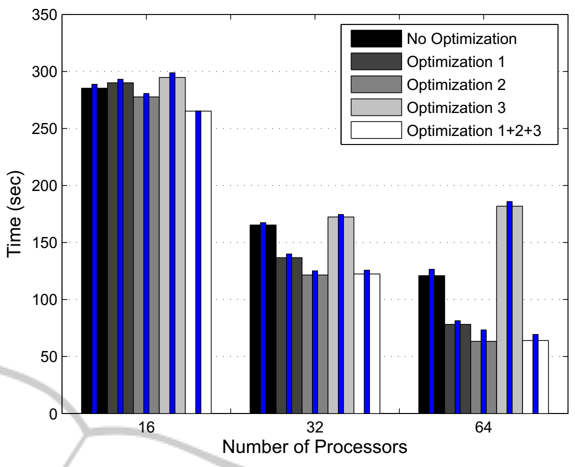

Figure 4: Scalability of 5 different optimization strategies on a 113-node cluster system. Regular bars represent SST/-macro DUMPI-driven simulation times with different optimizations and the core bars represent observed time.

4.2 Simulation of mpiBLAST Optimizations

To evaluate the various optimizations of mpiBLAST that we mentioned earlier in this section, we run SST/-macro and collected the simulation DUMPI traces using 16, 32, and 64 processors on the cluster system. Figure 4 shows the scalability and efficiency of each approach. The y-axis shows the total execution time in seconds for all processes of each approach, and the x-axis represents the number of processors that we used for our simulation runs. In our diagram we use the following notations: *Optimization 1* is the enabling of parallel output, *Optimization 2* uses virtual fragments, and *Optimization 3* pre-distributes database to workers before search begins. In addition, we also tested a version that includes all three mpiBLAST refinements named *Optimization 1+2+3*, and a version that excludes all optimizations named *No Optimizations*. The simulation results indicate that for our selected genome analysis, the sequence matching of the *Aedes aegypti* genome using sequences of size 1MB, Optimization 2: the use of virtual fragments provides the best scalability and efficiency. Optimization 2 leads to a speed-up of a factor of 2 or more compared to our No Optimization solution when executed on 64 nodes of our cluster system. This finding is not surprising given the exponential increase of the cost of accessing global memory with the increase of the participating compute nodes. Enabling the master process to collect and write output (Optimization 1) also led to a performance increase by about a factor of 2 on our 64 node execution. Our tests indicated that in all execution scenarios the use of static work pre-distribution alone (Optimization 3) led to a significant overhead in our genome sequencing analysis and slowed down the execution time. However, when combined with Optimization 1 and Optimization 2, static work pre-distribution did not lead to performance loss and even helped increase the speed of execution in certain scenarios. Enabling Optimization 2 helped in achieving faster execution in the tests we performed using 16 and 32 nodes, however the observed speed-up was not as significantly high as with the scenario with 64 nodes. Intuitively, this result demonstrates that Optimization 2 provides excellent scalability, however, due to the overhead of computing the fragments, the methodology is effective only when we have a system with a higher degree of parallelism. The graph in Figure 4 shows the same trend for Optimization 1, where enabling parallel output even deteriorated the execution time for the scenario of using 16 cluster nodes.

5 CONCLUSIONS

The application of hardware/software co-design has been a feature of embedded system designs for a long time. So far, hardware/software co-design techniques have found little application in the field of high-performance computing. The multi-core paradigm shift has left both software engineers and computer architects with a lot of challenging dilemmas. The application of hardware/software co-design for HPC systems will allow for a bi-directional optimization of design parameters where software specifications and behavior drive hardware design decisions and hardware constraints are better understood and accounted for in the implementation of effective application software. The use of cycle accurate simulation tools provides the data and insights to estimate the performance impact on an HPC applications when it is subjected to certain architectural constraints. In this work we demonstrated the application of a newly developed open-source cycle-accurate macroscale simulator (SST/macro) for the evaluation and optimization of data intensive genome sequence matching algorithms. We performed both trace-driven simulation and simulation based on application modeling. In our experimental set-up, we run an mpiBLAST sequence matching algorithm using 1MB sequences of the genome of the yellow fever mosquito, *Aedes aegypti*. Using this data intensive application as a canonical example, we validated the accuracy of SST/macro. In addition, the analysis of our performance data indicated that the use of dynamic data fragmentation leads to significant performance gains and high scalability on a distributed memory cluster system. The framework we have presented in this work allows for the evaluation and optimization of mpiBLAST application on a wide variety of platforms, ranging from a conventional workstation to a system allowing levels of parallelism that are not obtainable by existing supercomputers. This simulation ability can play a crucial role for the effective design and implementation of large-scale data intensive applications to be executed on the future multi-core hardware platforms, that often could include a wide variety of features including a heterogenous design of CPUs, GPUs, and even FPGAs. In our future work, we intend to further develop and distribute a full-scale SST/macro model implementation of the entire mpiBLAST library and make it available as a part of the SST/macro simulation distribution.

ACKNOWLEDGEMENTS

We express our gratitude to our team members of the SST/macroscale simulator team at Sandia National Laboratories, Livermore, CA: Scott Cranford, David Evensky, Joe Kenny, Nicole Lemaster, Jackson Mayo, and Ali Pinar. In addition, we thank Adrian Sandu from Virginia Tech and Philip Bell from the University of Central Florida, and the anonymous referees from SIMULTECH 2011 for providing helpful comments and suggestions.

REFERENCES

Adve, V., Bagrodia, R., Deelman, E., and Sakellariou, R. (2002). Compiler-optimized simulation of large-scale applications on high performance architectures. *Journal of Parallel and Distributed Computing*, 62(3):393–426.

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410.

Altschul, S., Madden, T., Schäffer, A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402.

Bjornson, R., Sherman, A., Weston, S., Willard, N., and Wing, J. (2002). TurboBLAST(r): A Parallel Implementation of BLAST Built on the TurboHub. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS'02)*, pages 183–190.

Braun, R., Pedretti, K., Casavant, T., Scheetz, T., Birkett, C., and Roberts, C. (2001). Parallelization of Local

BLAST Service on Workstation Clusters. *Future Generation Computer Systems*, 17(6):745–754.

Dally, W. and Towles, B. (2003). *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Darling, A., Carey, L., and Feng, W. (2003). The Design, Implementation, and Evaluation of mpiBLAST. In *Proceedings of ClusterWorld 2003*.

Emmert-Streib, F. and Dehmer, M. (2008). *Information Theory and Statistical Learning*. Springer Publishing Company, Incorporated.

Gardner, M., Feng, W., Archuleta, J., Lin, H., and Ma, X. (2006). Parallel Genomic Sequence-Searching on an Ad-Hoc Grid: Experiences, Lessons learned, and Implications. In *IEEE/ACM International Conference for High-Performance Computing, Networking, Storage and Analysis (SC'06)*.

Janssen, C., Adalsteinsson, H., Cranford, S., Kenny, J., Pinar, A., Evensky, D., and Mayo, J. (2010). A Simulator for Large-Scale Parallel Computer Architectures. *Inter. Jour. of Distributed Systems and Technologies*, 1(2):57–73.

Knüpfer, A., R.B., Brunst, H., Mix, H., and Nagel, W. (2006). Introducing the Open Trace Format (OTF). In Alexandrov, V., van Albada, G., Sloot, P., and Dongarra, J., editors, *Int. Conf. on Computational Science*, volume 3992 of *Lecture Notes in Computer Science*, pages 526–533. Springer.

Lin, H., Balaji, P., Poole, R., Sosa, C., Ma, X., and Feng, W. (2008). Massively Parallel Genomic Sequence Search on the Blue Gene/P Architecture. In *Proc. ACM/IEEE conference on Supercomputing (SC'08)*, pages 33:1–33:11, Piscataway, NJ, USA. IEEE Press.

Lin, H., Ma, X., Chandramohan, P., Geist, A., and Samatova, N. (2005). Efficient Data Access for Parallel BLAST. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS'05)*, page 72.2, Washington, DC, USA. IEEE Computer Society.

Lin, H., Ma, X., Feng, W., and Samatova, N. (2011). Coordinating Computation and I/O in Massively Parallel Sequence Search. *IEEE Transactions on Parallel and Distributed Systems*, 22(4):529–543.

Mathog, D. (2003). Parallel BLAST on split databases. *Bioinformatics*, 19(14):1865–1866.

Mintchev, S. and Getov, V. (1997). PMPI: High-Level Message Passing in Fortran 77 and C. In *Proc. Inter. Conference and Exhibition on High-Performance Computing and Networking (HPCN Europe '97)*, pages 603–614, London, UK. Springer-Verlag.

MPI (2009). MPI (Message Passing Interface) standards documents, errata, and archives of the MPI Forum. http://www.mpi-forum.org.

Prakash, S., Deelman, E., and Bagrodia, R. (2000). Asynchronous Parallel Simulation of Parallel Programs. *IEEE Transactions on Software Engineering*, 26(5):385–400.

Riesen, R. (2006). A Hybrid MPI Simulator. In *IEEE Inter. Conf. on Cluster Computing 2006*, pages 1–9.

Schuster, S. (2007). Next-generation sequencing transforms today's biology. *Nature Methods*, 5(1):16–18.

Sherwood, T., Perelman, E., and Hamerly, G. (2002). Automatically Characterizing Large Scale Program Behavior. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2002)*, pages 45–57.

SST/Macro (2011). SST: The Structural Simulation Toolkit, SST/macro the Macroscale Components, Open Source Release. http://sst.sandia.gov/about_sstmacro.html.

Stroustrup, B. (2000). *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Susukita, R., Ando, H., Aoyagi, M., Honda, H., Inadomi, Y., Inoue, K., Ishizuki, S., Kimura, Y., Komatsu, H., Kurokawa, M., Murakami, K. J., Shibamura, H., Yamamura, S., and Yu, Y. (2008). Performance prediction of large-scale parallell system and application using macro-level simulation. In *Proc. ACM/IEEE conference on Supercomputing SC '08*, pages 20:1–20:9, Piscataway, NJ, USA. IEEE Press.

Thorsen, O., Smith, B., Sosa, C., Jiang, K., Lin, H., Peters, A., and Feng, W. (2007). Parallel genomic sequence-search on a massively parallel system. In *Proc. Int. Conf. on Computing Frontiers (CF '07)*, pages 59–68, New York, NY, USA. ACM.

Top500 (2011). Top 500 SuperComputers Ranking at March 2011. http://www.top500.org.

Underwood, K. D., Levenhagen, M., and Rodrigues, A. (2007). Simulating Red Storm: Challenges and Successes in Building a System Simulation. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS'07)*, pages 1–10, Los Alamitos, CA, USA. IEEE Computer Society.

Vectorbase (2010). NIAID Bioinformatics Resource Center for Invertebrate Vectors of Human Pathogens. http://www.vectorbase.org.

Zheng, G., Wilmarth, T., Jagadishprasad, P., and Kalé, L. (2005). Simulation-based performance prediction for large parallel machines. *Int. Jour. Parallel Program.*, 33(2):183–207.