A System for Real-time Parallel Scientific Computation and Visualization using the IBM Blue Gene/L Supercomputer

Matthew Hamilton*, David Churchill†, R. Phillip Bording† and Kirk Jordan‡
* Department of Computer Science
† Department of Earth Science
Memorial University of Newfoundland
Email: {hamilton,davidc}@cs.mun.ca, pbording@mun.ca
‡IBM Deep Computing
Email: kjordan@us.ibm.com

Abstract— **The rapid growth of seismic, biological, and similar data sets has made computation and visualization an increasingly complex task. Due to the inherit architectural differences between supercomputing and visualization hardware, a system for real-time display of these data sets must be carefully designed. We present such a system, comprising all aspects of computation, communication, and visualization implemented on the IBM Blue Gene supercomputer, coupled with a grid of high resolution IBM T221 monitors. We use a socket-based communication scheme to pass computational output to the display, which is powered by IBM Deep Computing Visualization using Scalable Visual Networking. We illustrate this system using a 3-D elastic wave propogation model example.**

I. Introduction

Visualization is a excellent tool to aid scientists in exploring and interpreting data. Interactive visualization software enables researchers to create an effective cognitive representation of raw experimental or model data [1]. Key features and relationships present in the data can be isolated and disseminated to others in visual form, highlighting insightful information that could not be discerned otherwise due to its potentially enormous size and complexity.

Geoscientists exploring the features of the earth's crust are a typical example of those who can benefit greatly from scientific visualization tools. They typically make models that describe the movement of elastic and acousic waves in the earth's crust. Visualizing the model first can help researchers to verify its validity, as any abnormalities not immediately obvious in the mathematical description can be more directly seen on a screen. Assuming the correctness of the model, by stepping it through time and visualizing the resulting computations, geoscientists can detect features of the earth's crust by watching how the waves propagate in this medium.

For the purpose of this project, we solved an elastic wave propagation PDE using a staggered grid finite-difference method as described in [2] and [3]. This type of problem fits into a class of problems of a general form where we step a multidimensional volume forward in time, computing the values for the current volume from the results obtained in past volumes. This type of scheme responds well to parallelization- we simply assign each node in the parallel to a subvolume of the total volume. In our finite difference method, subvolumes that are adjacent are the only ones that need to pass messages during the course of the computation.

In order to compute a three-dimensional (3D) elastic wave propagation model of a size large enough to be useful to scientists, we would need an amount of memory approaching 88GB per shot with as many as 100,000 shots [4]. The time resouces we need are on the order of 10^{15} computations to complete one shot. The type of parallel computer that could solve such a problem in a reasonable time would be necesarily large and have enormous power requirements, impinging severely on the cost-effectiveness of computing this type of model.

Special supercomputing solutions such as IBM's Blue Gene/L (BG/L) have been specially designed to provide enormous computing space and time resources into a dense package with significantly reduced power requirements to that of standalone machines configured as a cluster. However, in trying to meet these special engineering requirements, such solutions are so specially designed that they can no longer easily interface with the machines designed to provide visualization requirements.

Given these problems, we must decouple visualization and computing hardware resources and devise a scheme to minimize data transfer between disparate parts of the system.

There is a further side issue of being able to present a high-enough resolution image to the user so that small features of the data can be detected in the context of the larger structures present in the data. Sophisticated visualization caves like the Landmark Visualization Laboratory at Memorial University are expensive and can be supplemented by high-quality flat-screen technology to expand visualization capability to the average user. Our solution to this problem is to use a parallel display of high resolution commodity grade monitors to achieve similar results.

The rest of the paper is organized as follows. In section II

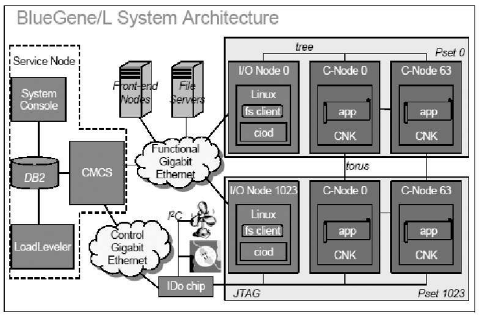

Fig. 1. Blue Gene/L System Architecture

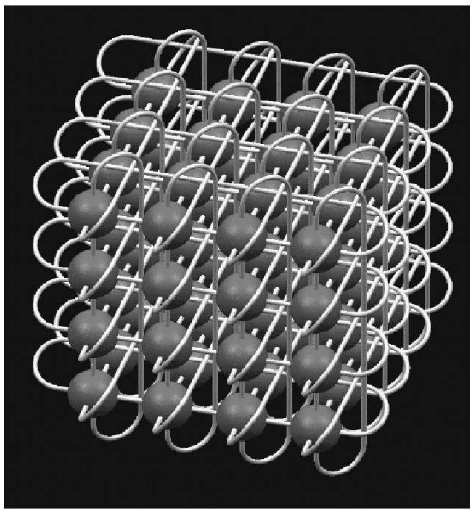

Fig. 2. Three dimensional torus network topology

we describe the various pieces of hardware which are needed to construct such a system. Section III talks about the software scheme we devised to run on each piece of hardware, enabling them to communicate efficiently. Finally, section IV details the conclusions we drew from this process, and outlines possible future work that can be done on the topic.

II. Hardware

A. *Blue Gene/L*

At 183500 GFlops peak performance the on Linpack benchmark, the 65,536 processor BG/L at DOE/NNSA/LLNL is the world's fastest supercomputer [5]. It was designed to yield low cost/performance levels of application-specific machines while remaining applicable to a broad range of applications amenable to massively parallel-based solutions [6].

Each compute node of the BG/L contains two PPC440 700MHz processors each capable of two simulataneous floating point operations per cycle. Though this is a moderate clock frequency, it enables many more processors to be operated within a small space, due to their low power consumption. One of these processors is by default intended to be an I/O coprocessor to offset the load incurred during I/O operations. However the two can also run in *virtual node mode*, whereby each processor handles its own communication, effectively doubling the number of compute node if I/O requirements are relatively low.

The compute nodes of the BG/L run a very restricted and stripped down unix kernel. It provides a very limited set of system call functionality, allowing only one process to run at a time. This kernel has been tuned to maximize performance of scientific computation. The BG/L system itself cannot provide the functionality to enable remote visualization as we desire.

The BG/L comes with a sophisticated set of interconnect networks. The collective and barrier networks have arithmetic and logic operations implemented in hardware along with a very low latency link intended for efficient execution of collective and barrier operations that make up a large part of many parallel computing applications.

The main interconnect network [7] allows point-to-point communication between any two nodes. The physical connect network is in the shape of a three-dimensional (3D) torus (Figure 2). This type of network makes best use of bandwidth when the parallel algorithms are restricted to local communication. In our case, the domain decomposition of the problem's 3D volume occurs in a way such that subvolumes that are adjacent in the total volume map onto nodes that are directy adjacent in the physical torus network of the BG/L. Furthermore, when-as in our case-the computional grid corresponds to a finite-difference solution to a PDE, the "wrapping" effect of the torus network maps nicely onto the local communication required for periodic boundary conditions.

Equally appealing about the BG/L system for our application is the set of highly optimized programming tools available. The dominant message passing model, MPI [8], has been implemented on the BG/L. Many of the BG's design parameters have been specially tuned to yield efficient performance with respect to MPI [6]. This in combination with IBM's XL Fortran and C/C++ compilers along with a standard programing library environment [9] should enable many existing high-performance computing applications to be ported to the BG environment with minimal difficulty. Our finite-difference fortran-based MPI code was almost effortlessly compiled and run on the BG system at the IBM Thomas J. Watson Research Center; in practice thus far we have had few portability issues.

Additonally, there has been some work done on developing performance monitoring tools which would be of great importance when we attempt to scale to more nodes while trying to identify bottlenecks and maximize system performance . The two papers [10] and [11] give an overview of what is already available and what is being developed by IBM research in this area.

B. *Visualization Hardware*

Due to the highly precise nature of scientific visualization, very high resolutions are needed in order to fully capture the nature of the simulation. In industry this is often done with either special purpose hardware, or using extremely expensive

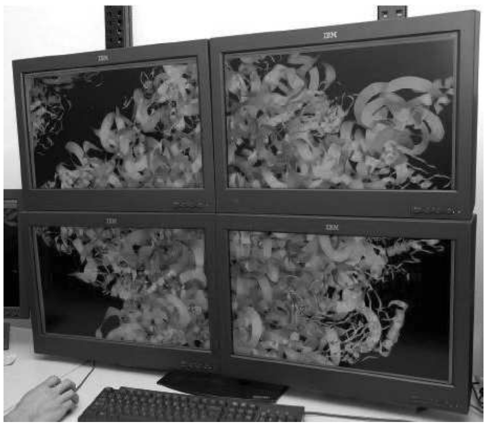

Fig. 3. A 2x2 wall of IBM T221 high resolution monitors

projector driven displays, which show the images on an extremely large scale, allowing researchers to see minute details. These type of systems however are out of reach for your typical researcher, so a more cost effective solution is needed. Deep Computing Visualization (DCV) and Scalable Visual Networking (SVN) from IBM [12] is a way for researchers to achieve this type of high resolution. SVN enhances images from visualization applications and allows them to be viewed in parallel on multiple, less expensive displays, achieving resolutions similar to larger theatre style visualization systems.

The hardware used to drive the visualization application of the system runs on a Linux machine using NVidia graphics cards. Using DCV and SVN, output from the OpenGL application is fed to four other Linux machines which each use their own NVidia graphics card to feed one of the four high resolution IBM T221 monitors, which combine to form a visualization wall. Each of these monitors has a maximum resolution of 3840x2400 for a total of 9.2 million pixels. With a 400 to 1 contrast ratio, each of these monitors provides an unparalleled view into scientific visualization, and when combined to form a 2x2 visualiztion wall, details never before available even to high end projector driven displays become crystal clear. This marriage of Linux, standard graphics cards, SVN, and parallel display on the T221 monitors make this system a truly cost effective way for researchers to visualize their data on a professional scale. Figure 3 above shows a biological visualization running on the the parallel 2x2 visualization wall.

III. THE SOFTWARE

A. *Finite-Difference Wave Propagation Code*

The acoustic and elastic wave equations are used here to simulate synthetic seismic shot records in 2D and 3D inhomogeneous media. These linear second order hyperbolic wave equations are set up using finite difference methods and a time marching scheme. This solution method is ideal for parallel computing using MPI and domain decomposition.

The code opens a socket to the front-end intermediary running off the BG/L and sends the results for a completed subvolume computation for each time step.

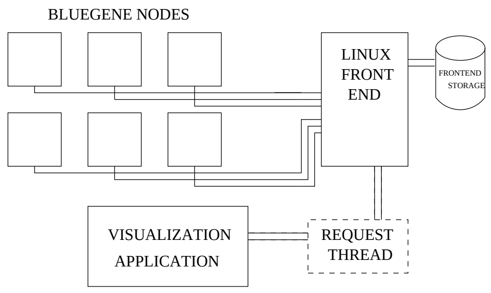

Fig. 4. Compute-Visualization System Diagram

B. *Challenge: Offline Visualization from Disk*

Our initial efforts in attempting to do straight-forward four-dimensional visualization of our data cube loaded all data from the completed computation into memory, then displayed this data on the screen. We quickly realized that the standard machines we had available to us had nowhere near the amount of physical memory necessary to store all of the data at once. To meet our goals of being able to quickly load the visualization, as well as provide an interactive manipulation of the data such as rotating camera angles, and the ability to time step through the computation showed that we would need a more clever way of selecting only the data we needed for the visualization to be accurate, and a quicker way of transferring it to the visualization machine. Simply letting the computation run to full, then copying the data file to another machine was too cumbersome a process to be useful. From this, the idea of socket based communication using an intermediary machine for the indexing of relevant data came to realization.

C. *The Front-end Intermediary*

The front-end intermediary comprises two distinct parts. The first of these parts recieves and stores subvolumes which have been computed by the compute nodes. Completed data is passed to the front-end by socket communication, as direct file I/O would create a bottleneck in getting data to the visualization client. As subvolumes are completed for a particular time step, they are sent to the front-end intermediary to be stored and indexed for later retrieval by the visualization software. Since our data is decomposed into possibly hundreds or thousands of subvolumes, this means that not all data is needed to be computed before the visualization application can start displaying data.

Any of these subvolumes which are completed and ready to be visualized can be sent to the visualization client via the second part of the front-end: the request thread. This request thread listens for requests from the visualization client about which subvolumes are needed to display current information. The requests are made by the visualization client by a caching and prefetching algorithm specific to the current type of data being displayed.

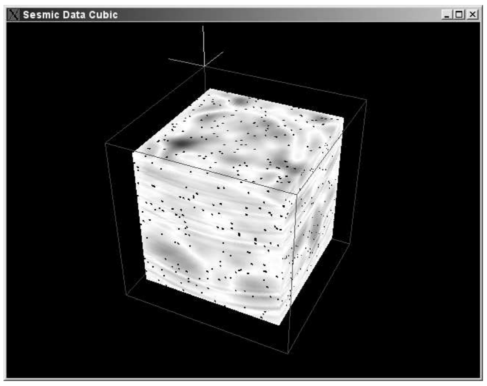

Fig. 5. Screenshot of the visualization client displaying partially computed data

D. Visualization Client

The visualization client application written in C, driven by the OpenGL graphics libraries, accepts data needed for display from an open socket to the front-end intermediary software.

Since various seismic models are quite different in nature, knowledge specific to the model itself must be known by the visualization client in order for prefetching and caching to be optimized. For example, our elastic wave propogation model took place in a data cube of size 200x200x200, because of this knowledge we knew that display of any three faces of the cube would be needed for a single time step in order for proper visualization to occur. Given the camera position, it is trivial to calculate which three faces of the cube are visible. For more complex visualization volume shapes, determining which parts of the volume are visible from given camera position is decidedly more difficult.

Caching took place based on spatial and temporal proximity - portions of the volume which are most likely to be viewed in the next frame of animation within the visualization client. This enables quick camera angle changes within the model to occur, as well as quick switching to either the next or previous time steps which had been computed. Since the requests sent to the frontend are for small domains of data which are being calculated in real time, we simply display the data which has been calculated, discarding the need for the entire model to be finished before anything is visualized. This allows us to achieve minimal memory usage on the visualization client side, preserving the speed and interactivity of the application. This enables researchers to visualize obvious properties or errors which may have occured in computation in the early stages of computation, saving possibly hours or days of compute time.

Figure 5 shows an illustration of the system working in real time. Several of the subvolumes of data which have not yet been computed are represented as black quadrilaterals, while the completed volumes are colored based on their computed values.

IV. Conclusions and Future Directions

It is a challenge even with today's high-performance computing technology to compute scientific models large enough to be useful to researchers. Large memory and compute time constraints are major problems, particularly when we want to try to visualize the data using hardware that is affordable to the average researcher.

Specialized computing solutions such as the BG/L allow us to compute such models fast enough at a lower costs than past efforts. Since the visualization end of our solution must be decoupled from the compute end, we devised a means to efficiently transfer the massive amounts of data needed by the visualization client.

Our initial efforts have been tested on a 32 node partition of an actual BG/L system. While these efforts were sucessful, other challenges may arise as we attempt to scale to thousands and tens of thousands of processors, such as message passing restraints and I/O bottenecking.

There is further work to be done investigating our prefetching and caching algorithms. As well, for more general visualization volume shapes, determining which parts of the volume are visible from given camera position is decidedly more difficult. If such a problem is theoretically intractable as we generalize to arbitrary sets of visualization volumes, perhaps efficient prefetching/caching algorithms implemented in specialized hardware may be needed in order to provide a responsive interactive visualization client to users.

Acknowledgment

This work was supported in part by an IBM Faculty Grant and the endowment provided by Husky Energy to Dr. Bording, who holds the Husky Energy Chair in Oil and Gas Research at Memorial University.

References

[1] D. H. Hepting, "A new paradigm for exploration in computer-aided visualization," Ph.D. dissertation, 1999, adviser-Robert D. Russell.

[2] J. Virieux, "Sh-wave propagation in heterogeneous media: Velocity-stress finite-difference method," *Geophysics*, vol. 49, pp. 1933–1942, 1984.

[3] ——, "P-sv wave propagation in heterogeneous media: Velocity-stress finite difference method," *Geophysics*, vol. 51, pp. 889–901, 1986.

[4] R. P. Bording and L. R. Lines, *Seismic Modeling with the Complete Wave Equations*, ser. SEG Course Notes Series, 1997, no. 8.

[5] "25th top500 list," in *International Supercomputer Conference (ISC2005)*, Heidelberg, Germany, 2005.

[6] G. Almsi, C. Archer, J. G. Castaos, J. A. Gunnels, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, K. Pinnow, J. Ratterman, B. D. Steinmacher-Burow, W. Gropp, and B. Toonen, "Design and implementation of message-passing services for the blue gene/l supercomputer," *IBM Journal of Research and Development*, vol. 49, pp. 393–406, 2005.

[7] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, "Blue gene/l torus interconnection network," *IBM Journal of Research and Development*, vol. 49, pp. 265–276, 2005.

[8] "Mpi: A message-passing interface standard," University of Tennesse, 1995, see http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html.

[9] G. L. Mullen-Schultz, *Blue Gene/L: Application Development*, first edition (draft) ed. IBM Red Books, March 2005.

[10] R. S. Germain, Y. Zhestkov, M. Eleftheriou, A. Rayshubskiy, F. Suits, T. J. C. Ward, and B. G. Fitch, "Early performance data on the blue matter molecular simulation framework," *IBM Journal of Research and Development*, vol. 49, no. 2/3, pp. 447–455, March/May 2005.

[11] X. Martorell, N. Smeds, R. Walkup, J. R. Brunheroto, G. Almsi, J. A. Gunnels, L. DeRose, J. Labarta, F. Escal, J. Gimnez, H. Servat, and J. E. Moreira, "Blue gene/l performance tools," *IBM Journal of Research and Development*, vol. 49, no. 2/3, pp. 407–424, March/May 2005.

[12] "Ibm servers deep computing solutions: Deep computing visualization," see http://www-03.ibm.com/servers/deepcomputing/visualization/.