Performance of Large-Scale Scientific Applications on the IBM ASCI Blue-Pacific System*

A. A. Mirin†

Abstract

The IBM ASCI Blue-Pacific System is a scalable, distributed/shared memory architecture designed to reach multi-teraflop performance. The IBM SP pieces together a large number of nodes, each having a modest number of processors. The system is designed to accommodate a mixed programming model and also supports a pure message-passing paradigm. We examine a number of applications on this architecture and evaluate their performance and scalability.

1 Introduction

The IBM ASCI Blue-Pacific System is a scalable, distributed/shared memory architecture designed to reach multi-teraflop performance. The system, which is located at Lawrence Livermore National Laboratory, is intended to meet the needs of the Accelerated Strategic Computing Initiative (ASCI) [1], whose purpose is to help ensure the safety and reliability of the US nuclear stockpile in the absence of testing. This presentation discusses the performance and impact of a three-dimensional hydrodynamics code, along with several other scientific applications, on the IBM-SP platform.

In Section 2 we describe the machine configuration for the ASCI Blue-Pacific system. Section 3 discusses programming and performance issues. In Section 4 we compare the performance of a hydrodynamics code when run with a pure message-passing model versus a distributed/shared memory programming model. In Section 5 we discuss performance and scalability of the sPPM hydrodynamics code. Section 6 covers other applications. Conclusions and future directions are presented in Section 7.

2 Machine Configuration

The IBM system consists of hundreds of shared memory processor (SMP) nodes, each node containing four PowerPC processors. Although future plans call for increasing the number of processors per node, that number is expected to remain modest compared to the number of nodes. This contrasts with the ASCI Blue-Mountain approach at Los Alamos National Laboratory, which involves a modest number of SGI Origin 2000 nodes, each containing a large number (up to 128 as of this writing) of processors.

The Blue-Pacific complex contains two components. By far the largest component, referred to as the Sustained Stewardship TeraOp (SST) system, is comprised of three 488-node sectors, with a total CPU count of 5856. Each node contains 1.5 to 2.5 GBytes of local

*This is LLNL Report UCRL-JC-131596. Work performed under the auspices of the U.S.D.O.E. by Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

†Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA

memory, and is powered by four 332 MHz PowerPC 604e processors. Allowing for up to 2 operations per clock period, the system peak performance is 3.9 TeraOPS. The processor to memory bandwidth is 2.1 Tbyte/s (aggregate), and the node to node bandwidth is 150 MByte/s (bidirectional). There are 62.5 TBytes of RAID storage, with an I/O bandwidth to local disk of 10.5 GBytes/s (aggregate). The three sectors are connected by six High Performance Gateway Node (HPGN) switches. As of this writing only two of the three 488-node sectors are in place. The smaller component, referred to as the Combined Technology Refresh (CTR), has much the same technology as the SST but only one sector containing 336 nodes.

3 Programming and Performance Issues

The IBM architecture accommodates itself to several programming models. An approach that is favored by many is the "mixed" programming model, in which one invokes shared memory parallelism on-node and message-passing between nodes. Common ways of attaining shared memory parallelism include Posix threads (p-threads), which are a standard within C, or the newer OpenMP [2], designed to accommodate Fortran codes (and more recently C codes as well). Message-passing between nodes is usually accomplished with Message Passing Interface (MPI). A second approach is to pretend that each processor has its own private memory and use MPI both across and within nodes.

There are various tradeoffs between the mixed model and the distributed memory, or "pure MPI" model. The latter is clearly simpler, because it invokes only one form of parallelism. It also accommodates distributed memory codes that run on other massively parallel processor architectures. The mixed model, on the other hand, offers greater freedom in how to parallelize and utilize memory within a node, and is likely to result in less communication overhead and fewer redundant computations.

This can be seen by looking at the very common example of two-dimensional domain decomposition. Given a square configuration with four available compute nodes, with a mixed programming model one is likely to assign a two-by-two domain decomposition. Using pure MPI it would be common to further subdivide each subdomain into four smaller subdomains using a two-by-two pattern. The resulting surface area to volume ratio (which measures the overhead of communication relative to computation) would then be twice as large as compared with the mixed model. Furthermore, with pure MPI there would likely be additional computations at the "inner" borders of the smaller subdomains.

With the distributed memory model, the on-node parallelism would be based strictly on the two-by-two domain decomposition within a node. Using the mixed model one could choose to parallelize along rows or columns or in some other fashion. Additional issues to be considered are light-weight threads versus full-blown processes, multiple versus single communication requests per node, and the implementation of on-node "communication".

A key issue in the ASCI program is scalability. This is not to be confused with the related concept of parallel efficiency. It is common to look at a fixed problem and see what happens as more and more processers are assigned to solve that problem. Scalability, however, deals with increasing the problem size while commensurately increasing the number of processors. Explicit finite-difference methods and multigrid methods are examples of scalable technology. Many algorithms in use today are not. Making effective use of hundreds to thousands of computational processors will require scalable algorithms.

An issue that affects scalability is load balance. An idle processor lowers the parallel efficiency and hence scalability of an algorithm. Problems that lend themselves to a regular decomposition have a better chance of being load balanced provided the computational work per degree of freedom (e.g., meshpoint) is uniform. Algorithms that make use of adaptive mesh refinement can be more difficult to load balance since the spatial variation of the workload is evolving in time. Having several times as many tasks as processors may increase the opportunity to balance the load, but possibly at the cost of more complex or increased communication.

Not only do we need the computation to scale favorably, but we need to have scalable I/O as well. Present-day high-resolution calculations are very often limited by the ability to get data off the machine and onto disk. With limitations on the bandwidth between disk and memory, having all computational processors interacting with disk is often less efficient than having just a few playing that role. Furthermore, not all systems even allow multiple processors simultaneously writing to disk. A mode of operation that is becoming more typical is for an application to dedicate one or more processors to I/O.

4 Parallel Programming Paradigm

We compare the pure MPI and mixed MPI/OpenMP programming models, when applied to the PPM hydrodynamics code [3]. The PPM code is a predecessor to the sPPM code, to be discussed in the following section. Of relevance here is that PPM is computationally intensive and contains primarily local communications.

The IBM system supports two forms of communication between nodes - IP and US. The IP communication library uses Internet Protocol for communication between nodes (and within a node as well), while the US (user space) communication library allows the application to communicate directly with the switch without going through the kernel or operating system. User Space mode is faster, but more restricted. In particular, the operating system on the machine at this writing supports User Space communication for only one processor at a time per node. Hence, when running pure MPI one must use the much slower Internet Protocol. By the time of this presentation we expect that a newer operating system that supports communication from multiple processors per node in User Space will have been implemented in the production environment. We expect to report on the programming paradigm comparison at that time.

5 The SPPM Hydrodynamics Code

The SPPM code [4] is a simplified version of the Piecewise Parabolic Method (PPM) Code mentioned above. It solves the single-fluid compressible Navier Stokes equations in three dimensions using the piecewise parabolic method, which is a higher-order accurate Godunov method developed by Colella and Woodward [5]. The code uses a Lagrangian time advance, followed by a remap onto the original grid, making the calculation effectively Eulerian. Multidimensional aspects are handled through operator splitting in the coordinate directions.

The code simultaneously exploits explicit threads for multiprocessing shared memory parallelism and domain decomposition with message passing for distributed memory parallelism. For each coordinate sweep a subdomain is partitioned into one-dimensional "pencils," within which data is optimally cached; the pencils are then assigned to the threads. The internodal communication is asynchronous, designed to overlap the computation.

We apply SPPM to study the interaction of a shock with a contact discontinuity[1]. The two fluids in contact are initally separated by a membrane adjacent to a wire mesh, as described in [6]. An initial perturbation containing both a long wavelength component, corresponding to the distortion of the mesh as a whole as it is being pushed, and a short wavelength component, corresponding to the mesh spacing, is applied. The interaction of the (Mach 1.5) shock with the interface leads to the well-known Richtmyer-Meshkov instability.

The simulation had over 8 billion zones and was carried out in single precision (32-bit arithmetic) using 960 nodes of the SST system. The global mesh had dimensions $2048 \times 2048 \times 1920$, organized according to an $8 \times 8 \times 15$ domain decomposition, resulting in a local (per node) mesh of $256 \times 256 \times 128$. The case was run for 27,000 timesteps, corresponding to 9 transverse sound crossing times. It took 173 hours of computer time, spread out over 226 hours of wall time. The computational throughput was estimated to be 129 Mflop/s per node, for an aggregate rate of 494 Gflop/s. The code executed at about 75 per cent of its peak performance, partially because of the inability to fully overlap communication and computation at this grid layout. Load imbalance was minimal.

The executable and all code output resided on local disk (the application had access to 4 GByte/node). Restart dumps (containing one component per node) were produced every three hours, and components were immediately copied onto alternate nodes as backups. Two hundred seventy four movie frame dumps, containing an aggregate 2.2 TByte of data spread out over 263,000 files, were produced. Ten 16-bit compressed data dumps, containing 84 GByte each, were produced as well. All together the application produced almost 300,000 files and 3 TByte of information. Data to be saved was copied onto the global file system, then to the visualization server, and finally to mass storage. Thus, the parallel I/O strategy was to write data to each local node and leave it to the postprocessor to gather the data coherently.

Output from the simulation shows the development of bubbles and spikes, along with their merger and breakup. Figure 1 shows a volume rendering of an entropy-like variable near the end of the simulation. We hope to evolve this simulation further in time and apply a second shock as well.

6 Other Applications

Here we describe additional scientific applications run on the IBM SST system at LLNL. Up-to-date results will be presented at the meeting.

6.1 First-Principles Molecular Dynamics

The JEEP code is a first-principles three-dimensional molecular dynamics code[2]. It computes the trajectories of atoms using forces calculated from quantum mechanics. The main approximation used in the solution of the Schroedinger equation is Density Functional Theory (DFT), in which the energy is assumed to be a unique functional of the electronic charge density. This approach allows one to describe accurately the formation and breaking of chemical bonds in a wide variety of conditions, thereby making this approach applicable to high temperature and high pressure physics, as well as biochemistry. JEEP is also used to study the properties of semiconductor clusters and surfaces.

[1] Investigators for this study are R. H. Cohen, B. C. Curtis, W. P. Dannevik, A. Dimits, M. A. Duchaineau, D. E. Eliason, A. A. Mirin, D. H. Porter, and D. R. Schikore.

[2] Investigators for this study are F. Gygi, G. Galli and F. Ree.

JEEP is based on the plane wave pseudo-potential approach to electronic structure. It relies heavily on an efficient implementation of linear algebra and Fast Fourier Transforms. It uses MPI across nodes and multithreading within a node.

Typical simulations consist of observing the time evolution of a few hundred atoms over several tenths of a picosecond. Using 960 nodes on the SST system, JEEP was able to simulate 600 atoms for approximately 1 picosecond. This type of breakthrough will allow simulation of complex liquids and of solvation of biomolecules in water.

6.2 Neutron Transport

The ARDRA code solves the Boltzmann transport equation for neutrons, which is an integro-differential equation in six-dimensional phase space[3] The code uses a Petrov-Galerken linear, continuous finite element discretization in space and either an S_n or a P_n discretization in energy [7]. The code uses MPI across nodes and Posix threads within a node.

ARDRA has been used to simulate the flux of fusion neutrons exiting the target chamber of the NOVA laser at LLNL. Because the neutrons can be very penetrating, it is important to know in which directions the neutrons will tend to emanate, so that appropriate shielding can be put in place. The ARDRA team has built a "numerical prototype" of the target chamber, accounting for the smallest structures present. Running on 960 nodes of the SST system, ARDRA has been able to execute using 160 million spatial dimensions, 4 moments, and 23 energy groups, for a total of 14.7 billion degrees of freedom. The ability to attain this level of detail and accuracy has been a significant contribution to the NOVA effort. Attempts are being made to develop a similar capability for other experiments as well.

6.3 Arbitrary Lagrange Eulerian Calculations

The ALE3D code uses the finite element method for treating fluid and elastic-plastic response on an unstructured, hexahedral grid[4]. The code uses an Arbitary Lagrange Eulerian (ALE) method with slide surface boundary conditions between mesh blocks [8]. ALE3D uses domain decomposition with MPI across nodes. It can be run using either pure MPI or OpenMP. Capability is being developed for treating implicit hydrodynamics, chemistry, and thermal radiation. The implicit and thermal physics will make use of the ISIS Finite Element Interface for the linear algebra.

ALE3D has been run on the SST system to simulate implosion in a cylindrical configuration. As the parallel programming methodology is evolving, the performance is in a rapid state of flux and will be presented at the meeting.

6.4 Quantum Chemistry Computations

The MPQC (Massively Parallel Quantum Chemistry) code computes the Hartree-Fock (HF) energy of molecules and augments it with second order Moeller-Plesset perturbation theory[5]. Most of the computational work is involved in computing electron repulsion integrals and transforming them to a different basis [9]. This involves the rearrangement of data on the nodes. The code is written entirely in C++ and uses the MPI library for

[3] Investigators for this study are P.N. Brown, B. Chang, U. Hanebutte, S. Smith, M. Dorr, R. Buck, J. Hall, S. Post, J. Ferguson, J. Rogers, P. Nowak, M. Zika and S. Hadjimarkos.

[4] Investigators for this study include W. S. Futral and R. Sharp.

[5] Investigators for this study are C. L. Janssen, I. Nielsen, E. Seidl and M. Colvin.

communications. A distributed/shared memory approach is implemented that invokes 7 threads per node, 4 of them being for computation.

Running on 400 nodes on the IBM SST, MPQC has recently calculated, using 1329 basis functions, the binding energy for the water trimer in less than 10 hours. This is the largest Moeller-Plesset calculation that has been done to date. Previously, a calculation of comparable fidelity could have been applied only to the level of complexity of the water dimer. Given the capability of the SST system, we expect to be able to treat a wide range of systems with unprecedented levels of accuracy.

7 Conclusions and Future Directions

We have discussed the performance of a three-dimensional hydrodynamics code, along with several other scientific applications, on the ASCI Blue-Pacific System. We have shown that a number of these applications do indeed scale to thousands of processors. At the time of the presentation we expect that all three sectors of the SST machine will be on site and working together, so that even more ambitious computations may be carried out.

References

[1] *Accelerated Strategic Computing Initiative*, World Wide Web http://www.llnl.gov/asci, Lawrence Livermore National Laboratory Report UCRL-MI-125923.

[2] *OpenMP*, World Wide Web http://www.openmp.org.

[3] A. A. Mirin, et al., *Three Dimensional Simulations of Compressible Turbulence on High-Performance Computing Systems*, Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis (1997).

[4] S. E. Anderson and P. R. Woodward, World Wide Web http://www.lcse.umn.edu/research/sppm, Laboratory for Computational Science and Engineering, University of Minnesota (1995).

[5] P. Colella and P. R. Woodward, *The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations*, J. Comput. Phys., 54 (1984), pp. 174-201.

[6] M. Vetter and B. Sturtevant, *Experiments on the Richtmyer-Meshkov Instability of an Air/SF_6 Interface*, Shock Waves 4 (1995), pp. 247–252.

[7] S. F. Ashby, P. N. Brown, M. R. Dorr and A. C. Hindmarsh, *A Linear Algebraic Analysis of Diffusion Synthetic Acceleration for the Boltzmann Transport Equation*, SIAM. Jour. Numer. Anal., Feb. 1995, pp. 128–178.

[8] C. J. Aro, E. I. Dube, W. S. Futral and J. D. Maltby, *Coupled Mechanical / Heat Transfer Simulation on MPP Platforms using a Finite Element / Linear Solver Interface*, Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio (1999).

[9] C. L. Janssen, E. T. Seidl and M. E. Colvin, *Object-Oriented Implementation of Parallel ab Initio Programs*, Parallel Computing in Computational Chemistry, ed. T. G. Mattson, ACS Symposium Series 592 (American Chemical Society, Washington, DC, 1995), p. 47.

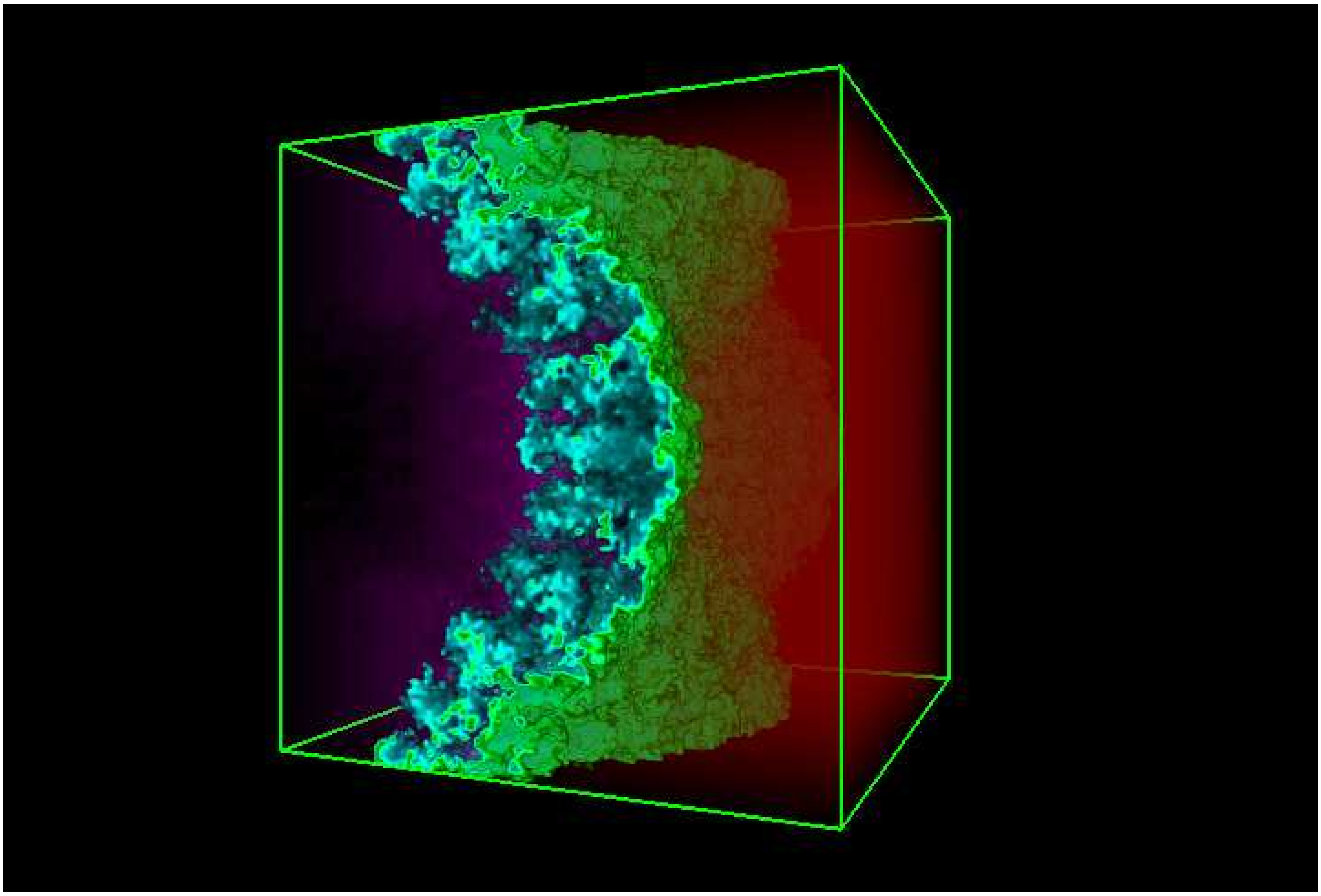

FIG. 1. *Volume rendering of entropy for sPPM calculation*