Tera-Scalable Algorithms for Variable-Density Elliptic Hydrodynamics with Spectral Accuracy

Andrew W. Cook, William H. Cabot, Michael L. Welcome*,
Peter L. Williams, Brian J. Miller, Bronis R. de Supinski
and Robert K. Yates

Lawrence Livermore National Laboratory
*Lawrence Berkeley National Laboratory

Abstract

We describe *Miranda*, a massively parallel spectral/compact solver for variable-density incompressible flow, including viscosity and species diffusivity effects. *Miranda* utilizes FFTs and band-diagonal matrix solvers to compute spatial derivatives to at least 10th-order accuracy. We have successfully ported this communication-intensive application to BlueGene/L and have explored both direct block parallel and transpose-based parallelization strategies for its implicit solvers. We have discovered a mapping strategy which results in virtually perfect scaling of the transpose method up to 65,536 processors of the BlueGene/L machine. Sustained global communication rates in *Miranda* typically run at 85% of the theoretical peak speed of the BlueGene/L torus network, while sustained communication plus computation speeds reach 2.76 TeraFLOPS. This effort represents the first time that a high-order variable-density incompressible flow solver with species diffusion has demonstrated sustained performance in the TeraFLOPS range.

SC|05 November 12-18, 2005, Seattle, Washington, USA (c) 2005 ACM 1-59593-061-2/05/0011...$5.00

1 Introduction

It is well known that low-order accurate solutions to the compressible Euler equations on parallel computers require only nearest neighbor communication and thus are easily parallelized (Cohen *et al.*, 2002). Solutions to the variable-density incompressible Navier-Stokes equations, however, are much more difficult to obtain on parallel computers due to their elliptic nature. Additionally, the inclusion of variable-density and diffusion effects further complicates the equations, making parallel simulations much more difficult, compared to the single fluid case. For a single fluid in a periodic box, it is natural to solve the equations in Fourier space, where FFTs only need to be performed on the nonlinear term (Yokokawa *et al.*, 2002). For the variable density case, with species diffusion and species-dependent viscosities, the equations contain variable coefficient fluxes as well as triple products; hence, the advantages of living in Fourier space are lost and many more terms appear, all of which require transforming.

If the flow is turbulent, i.e., possesses a wide range of length scales, then spectral and/or Padé (compact) methods are highly desirable, since they can accurately represent a broad range of wavenumbers (Lele, 1992). Spectral and compact methods involve implicit derivatives, thus further complicating the parallelization strategy. In this paper we demonstrate a scalable and efficient method for achieving high-order accurate solutions for variable-density viscous incompressible turbulence and compare our strategy to various alternatives. In particular, we consider incompressible Rayleigh-Taylor instability (RTI), since it exhibits all of the above-mentioned difficulties.

RTI is the baroclinic generation of vorticity at a perturbed interface subject to acceleration in a direction opposite the mean density gradient (Rayleigh, 1883; Taylor, 1950). The resulting interpenetration and mixing of materials has far-reaching consequences in many natural and man-made flows, ranging from supernovae to Inertial Confinement Fusion (ICF). In supernovae, the rate of growth of the mixing region is thought to be a controlling factor in the rate of formation of heavy elements. In ICF, accurate prediction of the depth of interpenetration of the fluids is crucial in designing capsules to maintain shell integrity.

We have investigated various parallelization strategies for computing RTI flows on the BlueGene/L (BGL) system. In particular, we have ported the *Miranda* code (Cook *et al.*, 2004) to this machine and have performed a series of scaling studies to determine the optimal approach to simulating RTI on tens of thousands of processors. In Section 2, the governing equations and solution techniques employed by *Miranda* are laid out. Simulation results on BGL are presented in Section 3. Section 4 contains a description of the BGL hardware and a discussion of key features which impact the scalability and performance of *Miranda's* algorithms. In Section 5, two parallelization strategies are compared, a direct block parallel matrix solution method and a transpose method. In Section 6, we attempt to optimize the transpose method on the BGL torus network. Section 7 contains scaling and performance data for *Miranda* using the transpose method. Finally, conclusions are presented in Section 8.

2 Miranda Code Description

Miranda solves the following equations for flows comprised of two incompressible miscible fluids in an accelerated Cartesian frame of reference:

$$\frac{\partial \rho}{\partial t} + u_j \frac{\partial \rho}{\partial x_j} = \rho \frac{\partial}{\partial x_j}\left(\frac{D}{\rho}\frac{\partial \rho}{\partial x_j}\right) \ , \tag{1}$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho g_i \ , \quad \tau_{ij} = \mu\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\right] \ ; \tag{2}$$

where ρ is density, $u_i = (u, v, w)$ is velocity, p is pressure, D is diffusivity, μ is viscosity and $g_i = (0, 0, -g)$ is acceleration. Spatial derivatives are computed in the code with the following 10th-order compact scheme

$$\beta f'_{j-2} + \alpha f'_{j-1} + f'_j + \alpha f'_{j+1} + \beta f'_{j+2} = c\frac{f_{j+3} - f_{j-3}}{6\Delta_i} + b\frac{f_{j+2} - f_{j-2}}{4\Delta_i} + a\frac{f_{j+1} - f_{j-1}}{2\Delta_i} \ , \tag{3}$$

$$\alpha = \frac{1}{2} \, , \ \beta = \frac{1}{20} \, , \ a = \frac{17}{12} \, , \ b = \frac{101}{150} \, , \ c = \frac{1}{100} \, ,$$

where j is a grid index along a line with N points in the i direction, and Δ_i is the grid spacing in that direction.

The solution is marched forward in time via the following 3rd-order variable-timestep predictor-corrector method. For equations of the form $\dot{\phi} = F(\phi)$, the predictor step is

$$\phi^* = \phi^n + \Delta t_{\text{new}} \left[(1 + \mathcal{R})F(\phi^n) - \mathcal{R}F(\phi^{n-1})\right] \tag{4}$$

and the corrector step is

$$\phi^{n+1} = \phi^* + \Delta t_{\text{new}} \left[\mathcal{A}F(\phi^*) + (\mathcal{B} - \mathcal{R} - 1)F(\phi^n) + (\mathcal{C} + \mathcal{R})F(\phi^{n-1})\right] \ , \tag{5}$$

where

$$\mathcal{R} = \Delta t_{\text{new}}/(2\Delta t_{\text{old}}) \ , \quad \mathcal{A} = (2\Delta t_{\text{new}} + 3\Delta t_{\text{old}})/[6(\Delta t_{\text{new}} + \Delta t_{\text{old}})]$$
$$\mathcal{B} = (\Delta t_{\text{new}} + 3\Delta t_{\text{old}})/(6\Delta t_{\text{old}}) \ , \quad \mathcal{C} = -\Delta t_{\text{new}}^2/[6\Delta t_{\text{old}}(\Delta t_{\text{new}} + \Delta t_{\text{old}})] \ ,$$

with Δt_{old} denoting the time increment between the $n-1$ and n timesteps, and Δt_{new} being the time increment between the n and $n+1$ times.

The density equation (1) is integrated by straightforward application of the predictor-corrector scheme. However, it must be advanced in conjunction with the momentum equation, which follows a pressure-projection algorithm. The pressure-projection scheme requires the solution of a Poisson equation. With periodic boundary conditions in x and y, the Poisson equation can be Fourier-transformed to obtain

$$\mathcal{F}_{xy}\left\{\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} = \Omega(x, y, z)\right\} \Rightarrow -k_x^2\hat{\hat{p}} - k_y^2\hat{\hat{p}} + \hat{\hat{p}}'' = \hat{\hat{\Omega}}(k_x, k_y, z) \ ,$$

where $\hat{\hat{p}}'' = \partial^2\hat{\hat{p}}/\partial z^2$ and Ω contains various space and time derivatives. Thus, with j being the z-index of the grid, $\hat{\hat{p}}''_j = \hat{\hat{\Omega}}_j + k^2\hat{\hat{p}}_j$, with $k^2 = k_x^2 + k_y^2$. An 8th-order compact approximation for $\hat{\hat{p}}''_j$ can be written as (Lele, 1992)

$$\beta\hat{\hat{p}}''_{j-2} + \alpha\hat{\hat{p}}''_{j-1} + \hat{\hat{p}}''_j + \alpha\hat{\hat{p}}''_{j+1} + \beta\hat{\hat{p}}''_{j+2} = b\frac{\hat{\hat{p}}_{j+2} - 2\hat{\hat{p}}_j + \hat{\hat{p}}_{j-2}}{4\Delta z^2} + a\frac{\hat{\hat{p}}_{j+1} - 2\hat{\hat{p}}_j + \hat{\hat{p}}_{j-1}}{\Delta z^2} \ , \qquad (6)$$

where $\alpha = 344/1179$, $\beta = 23/2358$, $a = 320/393$ and $b = 310/393$. The linear system for $\hat{\hat{p}}_j$ thus becomes

$$\left[\beta k^2 - \frac{b}{4\Delta z^2}\right]\hat{\hat{p}}_{j-2} + \left[\alpha k^2 - \frac{a}{\Delta z^2}\right]\hat{\hat{p}}_{j-1} + \left[k^2 + \frac{b}{2\Delta z^2} + \frac{2a}{\Delta z^2}\right]\hat{\hat{p}}_j$$

$$+ \left[\alpha k^2 - \frac{a}{\Delta z^2}\right]\hat{\hat{p}}_{j+1} + \left[\beta k^2 - \frac{b}{4\Delta z^2}\right]\hat{\hat{p}}_{j+2} = -\left[\beta\hat{\hat{\Omega}}_{j-2} + \alpha\hat{\hat{\Omega}}_{j-1} + \hat{\hat{\Omega}}_j + \alpha\hat{\hat{\Omega}}_{j+1} + \beta\hat{\hat{\Omega}}_{j+2}\right] \ , \qquad (7)$$

which is combined with Neumann boundary conditions and solved during both the predictor and corrector steps. Further details concerning the numerical scheme are described in Cook *et al.* (2004).

3 Simulation Results on BGL

The evolution of RTI is such that, at early times, the perturbations grow in a fairly independent fashion. Then the modes begin to couple to one another and secondary Kelvin-Helmholtz instabilities appear. At this point, the range of scales in the mixing layer rapidly increases, generating more mixed fluid within the layer. The large structures in the flow continue to increase until the mixing region becomes fully turbulent. Figure 1 is a snapshot of a RTI production run currently in progress on BGL. The simulation is running on 32,768 BGL nodes at a grid resolution of $3072 \times 3072 \times 1536$ points. The job size was chosen to fit comfortably into memory on the current machine. The computation will expand to 3072^3 grid points on 65,536 nodes once the second half of the machine arrives. About 56 hours of machine time (including I/O) has been used for the job thus far. At the current resolution and processor count, the code takes about 22 seconds to compute each timestep. Each restart dump requires about 8 minutes to read/write and utilizes 1.05 TB of disk space.

4 BlueGene/L Architecture

BGL was developed by IBM, in partnership with the Advanced Simulation and Computing program (ASC), as a massively-parallel computing system designed for research and development in computational sciences. Its goal is to deliver TeraFLOPS-scale computing on a routine basis to selected applications of interest to the U.S. Department of Energy's National Nuclear Security Agency. Its extremely high compute-density design results in a very

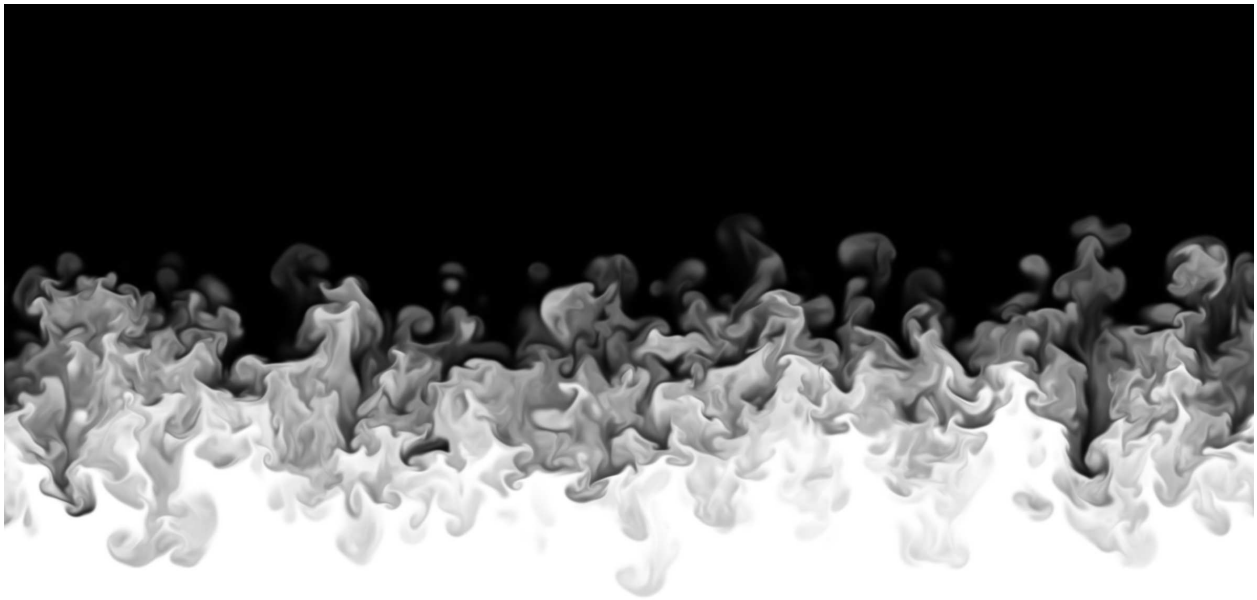

Figure 1: A two-dimensional slice from a three-dimensional Rayleigh-Taylor simulation on BGL. Light fluid (density=1) is white and heavy fluid (density=3) is black.

high cost-performance system with comparatively modest power and cooling requirements. At this writing, BGL is the fastest computer in the world, based on the 136.8 TFLOPS achieved on the LINPACK benchmark. The 32,768-node BGL system installed at LLNL will double in size to 65,536 nodes in late 2005. Here we provide only a high-level description of the system architecture, since BGL has been extensively described elsewhere (Adiga *et al.*, 2003; BGL, 2005).

A compute node of BGL is composed of 10 chips, a 700 MHz compute ASIC (Application-Specific Integrated Circuit) plus nine DRAM main memory chips. This highly integrated design drastically lowers power consumption and space requirements, while favoring communication and memory performance. The BGL chip is comprised of two independent PowerPC 440 cores, each capable of two floating point operations (FLOPs) per cycle (including fused multiply-adds, yielding a theoretical peak of 4 FLOPs per cycle per CPU), several independent network controllers, three levels of cache (including a 4 MiB L3), and memory controllers. Though each floating point unit is capable of two operations per cycle, the operations are not independent; i.e., the second floating point pipe is usable only by 2-way SIMD instructions or by 2-way SIMOMD (single instruction, multiple operation, multiple data) instructions (Bachega *et al.*, 2004). The theoretical peak of a single node is 5.6 GFLOPs,

hence 184 TFLOPs for the current 32,768-node system. The two processors on each chip are identical, with symmetric access to resources (but L1 cache coherence is not provided by the 440 core).

The system software supports two modes for applications to use the cores. In communication coprocessor mode there is a single MPI task per node, with one processor running the application and offloading much of the work of message passing to the second processor. In virtual node mode, each node runs two MPI tasks, one on each processor. MPI communications are handled by three independent special-purpose networks in BGL. Point-to-point and all-to-all communications are handled by a three-dimensional torus, with each node connected to its nearest neighbors via six independent bidirectional links. In addition to the torus, BGL has two tree-topology networks to perform global operations like broadcasts, reductions and barriers, with very low latency and high bandwidth. For example, an `MPI_Barrier` across 32,768 nodes is completed in under 2 microseconds. *Miranda* makes extensive use of all-to-all communications on various subsets of nodes, as discussed later. Thus, mapping the tasks onto the torus to reduce the overall communication time is one of the major challenges in running *Miranda* efficiently on BGL.

The BGL system software is under active development and new versions, with improved performance, are being installed on the system on a regular basis. Until recently, *Miranda* was unable to run in virtual node mode due to severe performance problems. As of software release level DRV202_2005, *Miranda* now works in virtual node mode with performance superior to coprocessor mode. Most of the timing studies reported herein were performed in coprocessor mode; however, the basic trends also apply to virtual node mode. Also, in Section 7 a complete set of scaling runs and performance numbers are reported for virtual node mode.

5 Parallelization Strategies for Band-diagonal Matrix Solvers

5.1 Communication Requirements

A core task of the *Miranda* code is computing implicit derivatives, f', from functions, f, using the compact scheme. Additionally, FFTs in the horizontal directions are required for the Poisson solve. The communications required for the matrix solvers and FFTs are similar; hence, our discussion focuses mainly on parallelization of the compact derivatives. Compact derivatives require solving the linear matrix problem

$$Af' = Bf \ . \tag{8}$$

For a 10th-order first-derivative, A is a pentadiagonal matrix and B is a heptadiagonal matrix (see Eq. 3). If all data for f in the given direction is contained on a process, the solution for f' is determined directly by a band-diagonal matrix solver based on LU decomposition. If f is distributed across some or all processors, then some communication overhead must be paid.

There are basically three strategies for solving band-diagonal linear systems (or computing Fourier transforms) in parallel: direct methods, whereby data are exchanged at processor boundaries then local solutions are obtained and joined back together; transpose methods, whereby the data are rearranged to give each processor all of the data it needs to compute a complete solution in serial; and iterative methods, whereby boundary data are exchanged and an initial guess is then iterated to convergence. Our preliminary tests of an iterative method showed it to be much more expensive than transpose or direct schemes; hence, our discussion here focuses on the latter two methods.

5.2 Domain Decomposition

For any of the above-mentioned strategies, a three dimensional computational domain can be broken up in any or all directions; e.g., a 3D mesh of size (n_x, n_y, n_z) can be distributed across a process grid of size (p_x, p_y, p_z), such that each MPI process contains a block of size $(a_x, a_y, a_z) = (n_x/p_x, n_y/p_y, n_z/p_z)$. The left-hand side of Figure 2 portrays a typical *Miranda* decomposition in which $p_z = 1$ and the process grid is decomposed into p_y X-communicator groups and p_x Y-communicator groups. These communicators are used to

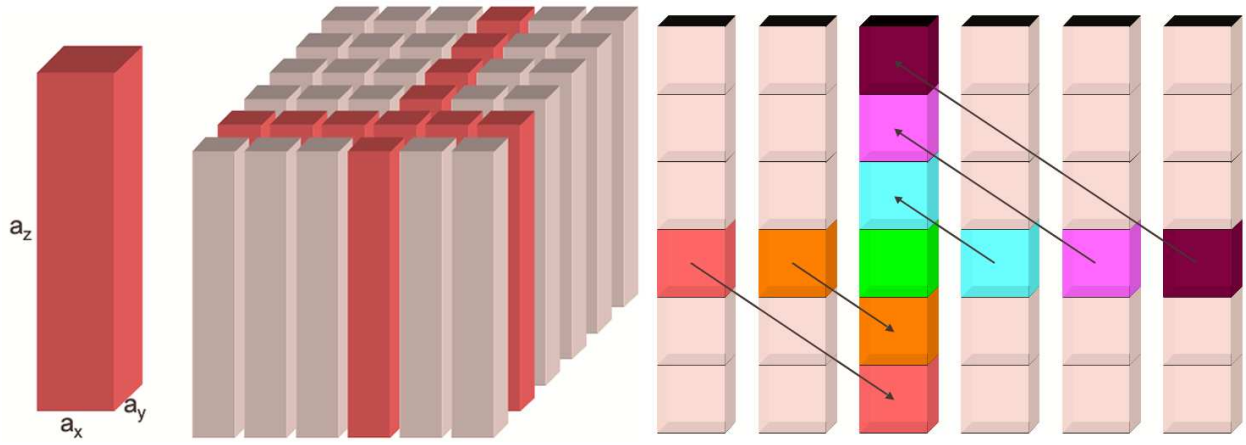

Figure 2: Left side: A sample two dimensional domain decomposition in *Miranda*. Right side: Block transfer in `MPI_Alltoall` operation for data transposes in *Miranda*.

martial data across processors for computing high-order spatial derivatives. The highlighted region on the left in Figure 2 shows a portion of the computational domain existing on a particular X and Y communicator.

5.3 Direct Approach to Solution

The direct method for solving pentadiagonal matrices in a direction of distributed data consists of a block parallel pentadiagonal solve (see e.g., Ivanov & Walshaw (2004)). In this method, local (incomplete) pentadiagonal solutions are performed on each process, boundary data is gathered across the communicator to form a global overlap solution, and then the

overlap solution is used by each process to complete the global solution. Computing the right-hand side of (8) requires sharing planes of boundary data with nearest neighbors; e.g., if B is heptadiagonal, 3 planes of data must be exchanged at each boundary with paired `MPI_Sendrecv` calls. The pentadiagonal A matrix on the left-hand side of (8) generates 4 planes (2 in each direction) of overlap data from the local solution. The global solution requires all overlap data, so `MPI_Allgather` is used to collect overlap data among processes. Each process then computes the global overlap solution independently (for load balancing) and completes the exact global solution. Note that the global overlap problem requires the solution of a 4×4 block-tridiagonal linear matrix problem with a dimension equal to the number of processes on the communicator (and hence is not strictly scalable). If the direction is globally periodic, the global overlap solution involves a periodic block-tridiagonal matrix.

5.4 Transpose Approach to Solution

In the transpose method, all the data along a pencil of cells in the desired coordinate direction must be collected onto a single process before the linear system is solved. For an X-Y domain decomposition, Z-derivatives require no communication, since all the required data is local to each process. For the computation of X and Y derivatives, `MPI_Alltoall` operations are used to reorganize the data, in a load-balanced manner, across all the processes within a communicator group. Consider, for example, the computation of an X-derivative. The data on each process can be logically decomposed into blocks of size $a_x a_y a_z / p_x$. The `MPI_Alltoall` operation acting on this data over the X-communicator group behaves like a data transpose operation. The jth data block from the ith process in the group is transported to the ith data block of the jth process in the group; this is illustrated by the graphic on the right-hand side of Figure 2. Following the `MPI_Alltoall`, local data reorganization is performed to align the data pencils at constant stride. The derivatives are computed in parallel, with each processor operating on its own block of data pencils. Another `MPI_Alltoall` operation sends the computed derivatives back to their home processors. An incentive for breaking up the data in X and Y, rather than Y and Z, is to minimimize the cost of the "descrambling" operation required after the `MPI_Alltoall` calls. With careful ordering of the data, the descrambling operations can performed as Fortran-intrinsic `transpose` and `reshape` calls on the first two (most contiguous) dimensions of the 3D arrays.

5.5 Comparisons

Various timings have been obtained on BGL for computing the gradient of a function,

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) , \tag{9}$$

using the 10th-order compact scheme (3). This is a common operation in *Miranda*, performed many times per cycle. Derivatives were computed with both the direct block parallel pentadiagonal ("BPP") scheme and with the serial pentadiagonal solves of globally transposed data ("XDP" scheme). The two methods were compared for both 2D ($p_z = 1$) and

3D domain decompositions. The runs were performed on the BGL hardware at LLNL in 512 through 32K node configurations, with a fixed problem size ($16 \times 16 \times 2048$ grid points) per node. The timings are presented in Table 1. All computations were performed in double

2D processor layouts		XDP times [s]				BPP times [s]			
processes	data per process	dx	dy	dz	grad	dx	dy	dz	grad
$16 \times 32 \times 1$	$16 \times 16 \times 2048$	0.17	0.17	0.03	0.37	0.57	0.65	0.03	1.25
$32 \times 32 \times 1$	$16 \times 16 \times 2048$	0.23	0.17	0.03	0.43	1.05	0.65	0.03	1.73
$32 \times 64 \times 1$	$16 \times 16 \times 2048$	0.34	0.16	0.03	0.53	0.60	1.06	0.03	1.69
$64 \times 64 \times 1$	$16 \times 16 \times 2048$	0.33	0.22	0.03	0.58	3.21	1.26	0.03	4.50
$128 \times 64 \times 1$	$16 \times 16 \times 2048$	0.32	0.17	0.03	0.52	7.76	4.48	0.03	12.27
$128 \times 128 \times 1$	$16 \times 16 \times 2048$	0.32	0.22	0.03	0.57	7.72	5.69	0.03	13.44
$256 \times 128 \times 1$	$16 \times 16 \times 2048$	0.31	0.22	0.03	0.56	9.76	4.55	0.03	14.34
3D processor layouts		XDP times [s]				BPP times [s]			
processes	data per process	dx	dy	dz	grad	dx	dy	dz	grad
$8 \times 4 \times 16$	$64 \times 64 \times 128$	0.18	0.18	0.18	0.54	0.14	0.08	0.10	0.33
$8 \times 8 \times 16$	$64 \times 64 \times 128$	0.24	0.18	0.18	0.60	0.16	0.11	0.10	0.37
$8 \times 8 \times 32$	$128 \times 64 \times 64$	0.34	0.17	0.18	0.69	0.14	0.11	0.17	0.43
$16 \times 16 \times 16$	$64 \times 64 \times 128$	0.33	0.21	0.18	0.72	0.22	0.16	0.11	0.49
$32 \times 8 \times 32$	$64 \times 128 \times 64$	0.31	0.33	0.18	0.81	0.24	0.09	0.17	0.50
$32 \times 32 \times 16$	$64 \times 128 \times 64$	0.31	0.31	0.24	0.86	0.24	0.19	0.10	0.52
$32 \times 32 \times 32$	$128 \times 64 \times 64$	0.32	0.33	0.34	0.99	0.17	0.18	0.15	0.50

Table 1: Mean time per gradient operation using the transpose pentadiagonal (XDP) scheme and the direct block parallel pentadiagonal (BPP) scheme on BGL with `-O3` optimization.

(8-byte) precision with `-O3` optimization. For this comparison, the machine was configured in communication coprocessor mode (one CPU task per node). For virtual node mode (two CPU tasks per node), timings are typically about 21% faster. The XDP scheme performs best for two-dimensional domain decompositions, because no communication is required for directions containing complete columns of data. The BPP method, on the other hand, performs best for three-dimensional decompositions, because a cubic data decomposition minimizes the surface-to-volume ratio of the grid blocks and a cubic processor distribution reduces the cost of the global overlap solution by minimizing the number of processes on each communicator. It is somewhat surprising that, despite the very different communication patterns, the XDP and BPP times are comparable to one another and have similar weak scaling properties. In Section 6, we explore the possibility of reducing XDP times via custom torus mappings.

6 Communication Efficiency

6.1 The Mapping Problem

For the transpose method, *Miranda* spends $40 - 65\%$ (depending on machine mode) of its runtime in `MPI_Alltoall` communication operations. Consequently, we have investigated methods for reducing this time by employing special mappings of the *Miranda* processes to the processors in the BGL torus. We have also investigated the efficiency of the current IBM `MPI_Alltoall` implementation by comparing it to an estimate of the minimum time required to complete such an operation.

On BGL, `MPI_Alltoall` communication is performed over the torus network. On this network topology, communication performance is improved by packing together the processes that communicate most frequently, in order to minimize hop distance and maximize the number of torus links available for communication. Given that `MPI_Alltoall` operations perform an equal amount of communication between every pair of processes in the communicator group, it is desirable that these processes map to nearby processors. For two-dimensional domain decompositions, finding a good mapping is complicated by the fact that each process, p, belongs to two distinct communicator groups, $X(p)$ and $Y(p)$, such that $X(p) \cap Y(p) = \{p\}$. It is difficult to construct a mapping that will closely pack the processes of $X(p)$ and $Y(p)$ for every process, p. We have experimented with various mappings that optimize the placement of the processes in the X communicator group, leaving the processes of the Y communicator groups to fall where they may. We are in the process of investigating other mapping techniques, including the use of space filling curves and simulated annealing. One advantage to the maps we currently generate is that they are regular in shape and tile the space of processors. This results in uniform communication times across all communicator groups, thus preventing communication load imbalance.

By default, processes are mapped by `MPI_COMM_WORLD` rank order to processors in the torus network in XYZ order. That is, if (q_x, q_y, q_z) is the size of the BGL partition, then the first q_x processes are mapped to locations $(0 : q_x - 1, 0, 0)$, the second q_x processes are mapped to $(0 : q_x - 1, 1, 0)$, etc. Alternatively, we can specify a mapping file that explicitly states the torus location for each process. *Miranda* constructs an X-Y process grid of size $p_x \times p_y$ via the MPI Cartesian communicator constructors. The result is that the first p_x processes in `MPI_COMM_WORLD` constitute the first X-communicator group. The second p_x processes form the second X-communicator group, and so on. By default then, each X-communicator group is mapped to contiguous X-direction rows of the BGL torus network. Consider a 16K processor configuration of shape $(16, 32, 32)$; by default, *Miranda* constructs a $(128, 128)$ process grid which maps the first X-communicator group onto the first 8 X-direction rows of the torus at locations $(0 : 15, 0 : 7, 0)$, the second group to $(0 : 15, 8 : 15, 0)$, etc. This results in the first Y-communicator group being mapped to the locations $\{(0, \alpha, 0 : 31), \alpha = 0, 8, 16, 24\}$ Consequently, the X-communicator groups are packed into subregions of shape $16 \times 8 \times 1$ whereas the Y-communicator groups are mapped to 4 Z-direction pencils of processors, each of length 32, distributed a distance of 7 hops apart in the Y-direction. Alternatively, we could produce a mapping that packs the X-communicator groups into a $16 \times 4 \times 2$ block, or $8 \times 4 \times 4$. Tightly packing the X-communicator groups

can improve X-direction communication, but at the expense of dispersing the processes in the Y-communicator groups.

In virtual node mode, each node contains two MPI processes, each with half the data of a corresponding coprocessor mode calculation. When packing the X-communicator groups onto blocks of nodes, it is important to ensure that both processes on each node belong to the same X communicator group. Also, since each process contains half the data as in coprocessor mode, and since both processes on a node share a common network interface, the volume of traffic on and off the node is nearly identical to that of coprocessor mode. The only difference is that the communication between the two processes on a node is performed through a memory interface rather than via network packets.

6.2 `MPI_Alltoall` Performance

We can construct a lower bound on the time required to complete an `MPI_Alltoall` operation on a mesh or torus network provided we make the assumption that the communication time is network bandwidth limited. The BGL torus network has six links on each node $(\pm X, \pm Y, \pm Z)$, each operating at a peak bandwidth, b, of 175 MB/sec. If we can compute the number of messages, m, of size s that traverse the most heavily loaded link in the torus, then the minimum time required to complete the communication operation is $t_{\min} = sm/b$. In general, it is difficult to determine the most heavily loaded link and the amount of data that it transports because messages in the BGL torus are broken into 256 byte packets which are adaptively routed; however, we can estimate this value.

Consider the two-dimensional mesh or torus as shown in Figure 3. The circles represent nodes and the lines represent communication links. The dashed lines represent links in a torus that are not present in a mesh (the Y-direction dashed links are not shown). Let

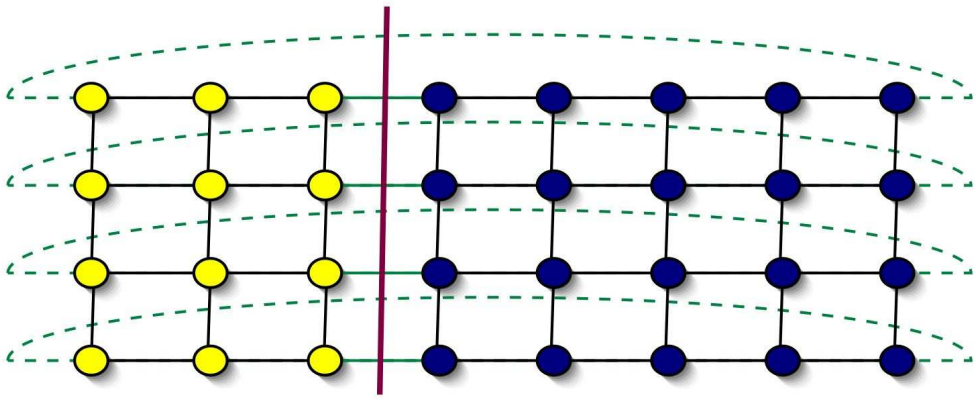

Figure 3: A two-dimensional processor mesh or torus network.

$\lambda_x = 1$ if the network is a mesh in the X direction and $\lambda_x = 2$ if it is a torus and similarly for λ_y; hence, λ compensates exactly for the additional path that the torus connection provides

between two colinear nodes. There are q_x processors in the X direction and q_y in the Y direction. We consider a cut through the $q_y\lambda_x$ X-links as shown, dividing the processors into left and right sets L and R. Let α be the number of nodes in L. The number of nodes in R is $q_xq_y - \alpha$ thus, the number of messages sent from L to R is $m(\alpha) = \alpha(q_xq_y - \alpha)$. This function has maximum $\alpha = q_xq_y/2$, stating that the set of links that carry the maximal message traffic are those that, if cut, would divide the network into two equal parts. The number of messages across this set of links is: $m_{\max} = q_x^2q_y^2/4$. We estimate the traffic across the maximally loaded link by taking the average of this message count across all cut links, or $m_{x,\max} = q_x^2q_y/(4\lambda_x)$. A similar argument for cuts in the X-direction yield an estimate of the maximal message count for any Y-direction link as $m_{y,\max} = q_y^2q_x/(4\lambda_y)$. So, an approximation of the minimum time required to complete an `MPI_Alltoall` operation on this 2D torus would be determined by the larger of these two values, or

$$t_{\min} = \mathrm{Max}\Big\{\frac{q_x}{\lambda_x}, \frac{q_y}{\lambda_y}\Big\}\frac{sq_xq_y}{4b} \ . \tag{10}$$

The extension to three dimensions is straightforward and yields the estimate

$$t_{\min} = \mathrm{Max}\Big\{\frac{q_x}{\lambda_x}, \frac{q_y}{\lambda_y}, \frac{q_z}{\lambda_z}\Big\}\frac{sq_xq_yq_z}{4b} \ . \tag{11}$$

Furthermore, we can extend this argument to `MPI_Alltoall` operations on subcommunicators within a rectangular region of shape $R = (r_x, r_y, r_z)$ provided the processes of the subcommunicators are uniformly distributed within this region. We can force the region to be minimal, in that it is the smallest rectangle that contains all processs in the communicator groups and that if $q \in R$ then all processes in the communicator group containing q are also in R. Let κ be the number of communicator groups in R, then we estimate the minimum communication time for all-to-all operations that execute simultaneously on all communicator groups within R to be

$$t_{\min} = \mathrm{Max}\Big\{\frac{r_x}{\lambda_x}, \frac{r_y}{\lambda_y}, \frac{r_z}{\lambda_z}\Big\}\frac{sr_xr_yr_z}{4b\kappa} \ . \tag{12}$$

$\lambda_x = 1$ if the BGL partition is a mesh in the X-direction or if $r_x \le q_x/2$. $\lambda_x = 2$ if the BGL partition is a torus in the X-direction and $r_x = q_x$.

In order to gain a better picture of how torus mappings affect communication efficiency, we collected timing data for *Miranda* runs in coprocessor mode on node partitions up to 32K. For these runs, we constructed a collection of process maps that pack the X-communicator groups into compact regions and timed the X and Y direction `MPI_Alltoall` operations. Figure 4 shows a comparison of these timings with the estimated minimum time from (12). Each set of bars in the figure shows timings for different mappings of the X-communicator groups for 512 through 32K nodes in coprocessor mode. For example, the $4 \times 4 \times 4$ bar of the 4K set indicates that the 64 processes of each X-communicator are mapped into a $4 \times 4 \times 4$ block of processors. The mapping corresponding to the default process layout is at the bottom of each set. The four segments of each bar are (from left to right): the estimated minimum time for an X-communictor `MPI_Alltoall` (blue), the measured X-communicator `MPI_Alltoall` time minus the minimum estimate (red), the estimated minimum time for

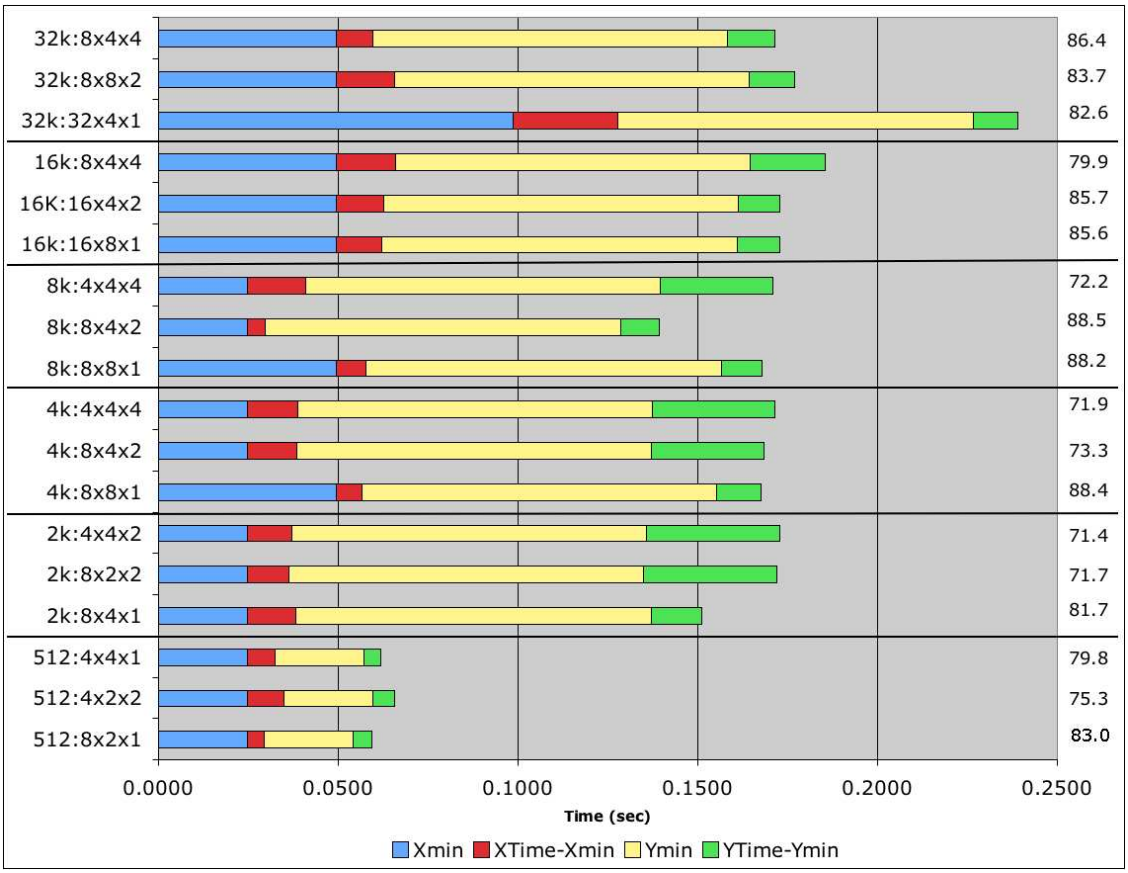

Figure 4: `MPI_Alltoall` communication timings for *Miranda* using various torus mappings on BlueGene/L.

the Y communicator (yellow) and the measured Y-communicator `MPI_Alltoall` time minus the estimated minimum (green). That is, the actual measured X communicator time is the sum of the blue and red bars and the actual Y communicator time is the sum of the yellow and green bars. Finally, when computing the estimated minimum `MPI_Alltoall` time via (12), we used a peak per link bandwidth of 170MB/sec rather than 175. This conservatively compensates for the overhead of MPI message and packet headers that are sent in addition to the actual data payload, since `MPI` send-receive pingpong bandwidth is approximately 150 MB/s. The values in the right margin of the chart show the communication efficiency of the mapping (as a percentage) with respect to the estimated minimum time. Figure 4 shows that communication times can vary significantly depending on how the MPI tasks are mapped to the BGL torus. Furthermore, it is clear that in the cases of 8K and 32K partitions, custom mapping files exist that siginificantly outperform the default maps. Finally, for partitions larger than 2K, `MPI_Alltoall` performance exceeds 85 percent of the theoretical peak speed of the network.

7 Overall Performance

7.1 Scaling

We now turn to the overall performance (communication plus computation) of *Miranda* using the tuned XDP scheme and simulating RTI on up to 65,536 processors of BGL. For the scaling studies, *Miranda* was compiled with IBM's **xlf90** Fortran compiler (version 9.1 for BGL), with optimization flags `-O3 -qalias=noaryovrlp -qmaxmem=-1 -qalign=4k -qhot=novector -qarch=440 -qtune=440 -qessl`. The scaling characteristics of *Miranda* in coprocessor mode and virtual node mode are essentially identical, with virtual mode being about 21% faster; therefore, we present only the virtual node mode results.

Weak scaling results (fixed workload per node) are shown in Fig. 5 and strong scaling

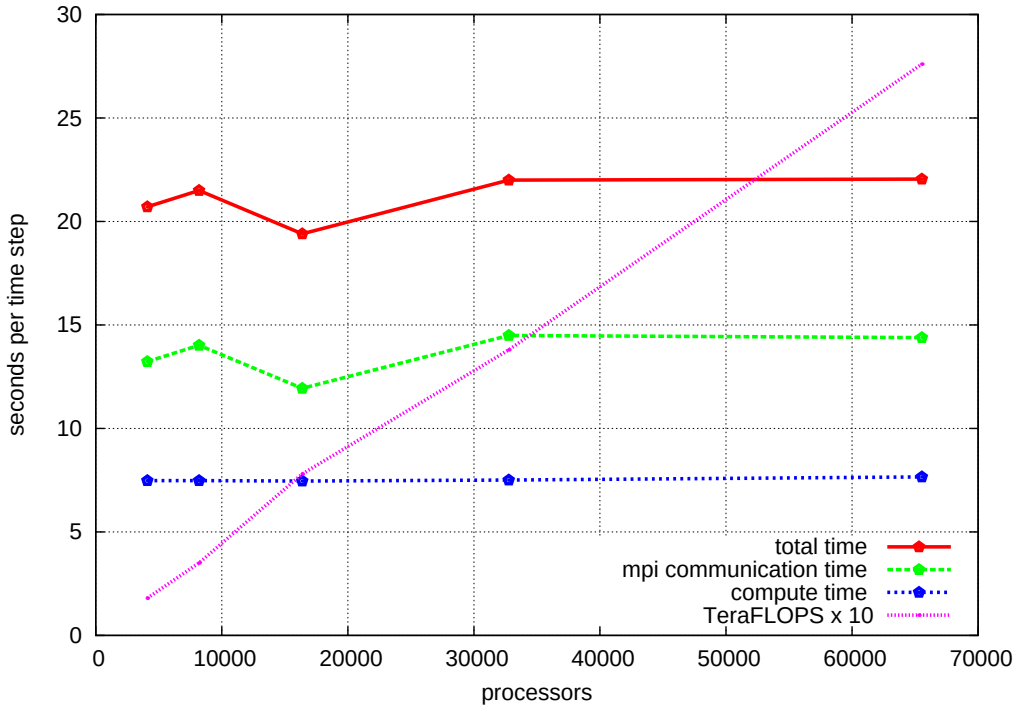

Figure 5: Weak scaling for *Miranda* using $8 \times 16 \times 2048$ grid points per processor.

results (fixed problem size) in Fig. 6. For the weak scaling runs, each processor contained a $8 \times 16 \times 2048$ point grid. For the strong scaling runs, $2048 \times 2048 \times 512$ total grid points were used. Custom torus mappings were employed to improve performance on 16K and 64K CPU partitions. From Fig. 5, it can be seen that the transpose approach to computing implicit derivatives yields near perfect scaling on the BGL architecture. Furthermore, the TeraFLOPS increase linearly as the problem is scaled up. Additional evidence for linear scaling is seen in Fig. 6, which shows near perfect speedup for a fixed problem size as more processors are added. The timings are shown relative to 8K processors because that is the smallest configuration on which the $2048 \times 2048 \times 512$ point grid would fit. The better-than-

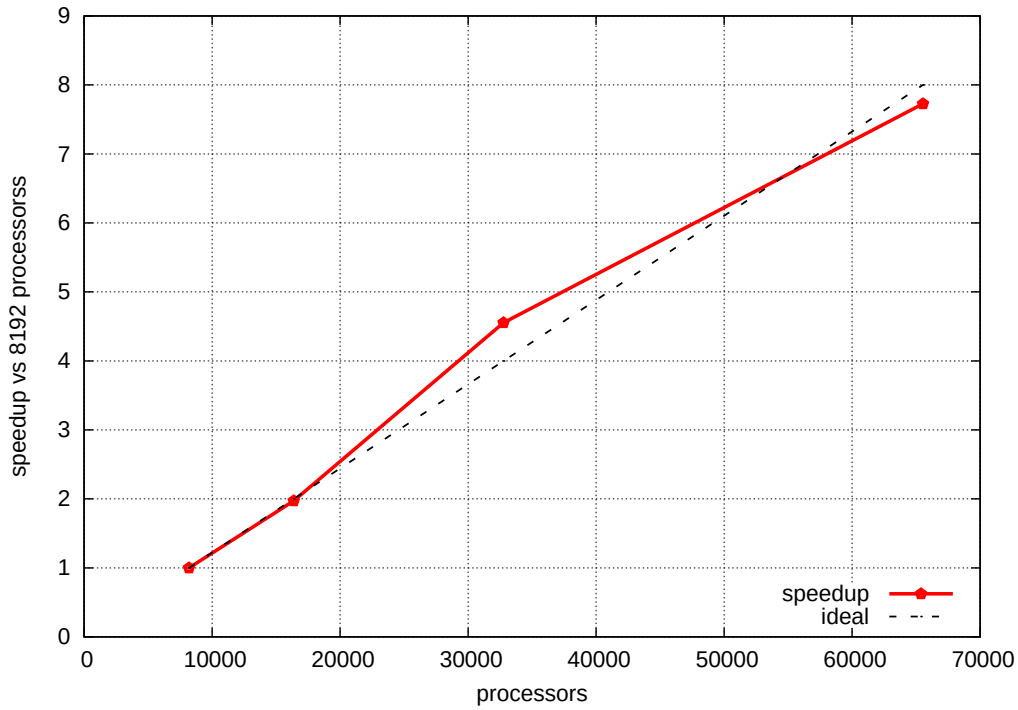

Figure 6: Strong scaling for *Miranda* using a $2048 \times 2048 \times 512$ point grid.

ideal speedup from 16K to 32K processors may be due to smaller message sizes or better cache use or both.

7.2 Tuning and FLOPS

In performing Raleigh-Taylor instability simulations with compact derivatives, *Miranda* spends significant time in the backsubstitution portion of the pentadiagonal matrix solver. We have focused our optimization efforts on this code section to improve floating point performance. Each processor must perform the solves on a block of independent data pencils. The all-to-all transposes in *Miranda* are written to allow matrix operations to be performed on any one of the 3 indices of an array. We choose the index for matrix operations to be the second, or J index, with the effect that computations in the inner I loops are independent. The BGL floating point units have single cycle throughput but latencies of 4 cycles for loads, 3 for stores and 5 for most arithmetic operations. It is therefore important to organize the code such that multiple independent floating point operations can be issued at any point in time. We provide this by unrolling the inner loops by a factor of 4. Beyond a factor of four, we found that the performance did not increase, possibly due to memory bandwidth limits or register pressure. Furthermore, we re-organized the matrix solver to provide temporary accumulation arrays, such that memory access occurs with a constant stride of one for each data stream. This fully engages the BGL Level 2 prefetch engine, reducing L1 cache misses. These optimizations improved the performance of the backsolves by a factor of 2.3.

Unfortunately, we have thus far been unable to realize any performance improvement by utilizing the second FPU on each CPU. For Miranda, the dual FPU looks like a two-element vector unit and the inner loops of the backsolve ought to vectorize trivially. All the arrays are 16-byte aligned and we have explicitly told the compiler this information via "alignx" calls, so that the requirements for quadword loads are satisfied. However, with the current Fortran compiler, dual FPU performance is worse than if the second FPU is ignored. We are awaiting compiler improvements, which ought to enable *Miranda* to make profitable use of the second FPU.

In addition to the virtual node mode tests, we also implemented a "dual core" version of the backsolver in coprocessor mode. This involved the use of coroutines and explicit L1 cache management techniques to export half of the backsolve work to the FPU on the communication coprocessor. Miranda is highly bulk-synchronous in that it is either doing communication or computation exclusively. During the backsolves, the communication coprocessor is idle and thus available to compute half of the workload without introducing load imbalance. Implementation of the dual core routines produced a 25% performance increase for the backsolves alone; however, overall performance fell short of the 21% increase obtained in virtual node mode; we have therefore abandoned this approach.

When profiling the entire code, the **BGL_perfcntr** library reports a sustained computing rate of 2.76 TFLOPS for *Miranda* running on 65,536 CPUs (in virtual node mode), with the core of the matrix solver operating at an aggregate 15.67 TFLOPS. Although codes with simpler physics and local differencing algorithms have achieved higher numbers, this level of performance is unprecedented for variable-density incompressible flow using compact differencing schemes.

8 Conclusions

The combined effects of variable density and diffusion on incompressible turbulent flows, coupled with the need for high fidelity numerical methods, presents a challenging problem for parallel computing on tens of thousands of processors. We have demonstrated that, contrary to popular opinion, transpose techniques for implicit derivatives can provide excellent scaling to such large numbers of processors and are highly competitive with direct block parallel matrix decomposition schemes. After porting the *Miranda* code to the BGL architecture and removing various memory and CPU bottlenecks, we have obtained nearly perfect weak scaling and linear speedup of the code to 65,536 processors. By packing the data tightly in one direction, we have reduced the communication overhead on 64K processors by 30% over the default mapping. We have also reached data transfer rates exceeding 85% of the theoretical peak of the network. In addition to optimizing the torus mapping for `MPI_Alltoall` communications, efforts are currently underway to tune the packet injection rate of the messages. Despite the substantial communication and data rearrangement requirements of the implicit 10th-order compact scheme, our initial efforts have achieved a sustained computational rate of 2.76 TFLOPS for *Miranda*, simulating RTI on 65,536 processors of the BGL machine.

Acknowledgments

We are grateful to Jim Sexton and Bob Walkup of IBM for their assistance in gathering timing information. This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

2005 BlueGene/L. http://www.llnl.gov/asci/platforms/bluegenel/.

Adiga, N. R. *et al.* 2003 An overview of the BlueGene/L supercomputer. SC2003: Supercomputing Conference.

Bachega, L., Chatterjee, S., Dockser, K., Gunnels, J., Gupta, M., Gustavson, F., Lapkowski, C., Liu, G., Mendell, M., Wait, C. & Ward, T. J. C. 2004 A high-performance SIMD floating point unit design for BlueGene/L: Architecture, compilation, and algorithm design. Parallel Architecture and Compilation Techniques Conference (PACT 2004).

Cohen, R. H., Dannevik, W. P., M., D. A., Eliason, D. E., Mirin, A. A., Zhou, Y., Porter, D. H. & Woodward, P. R. 2002 Three-dimensional simulation of a Richtmyer-Meshkov instability with a two-scale initial perturbation. *Phys. Fluids* **14**, 3692–3709.

Cook, A. W., Cabot, W. & Miller, P. L. 2004 The mixing transition in Rayleigh-Taylor instability. *J. Fluid Mech.* **511**, 333–362.

Ivanov, I. G. & Walshaw, C. 2004 A parallel method for solving pentadiagonal systems of linear equations. University of Greenwich Report, http://staffweb.cms.gre.ac.uk/∼c.walshaw/papers/fulltext/IvanovTR3698.ps.

Lele, S. K. 1992 Compact finite difference schemes with spectral-like resolution. *J. Comput. Phys.* **103**, 16–42.

Rayleigh, L. 1883 Investigation of the character of the equilibrium of an incompressible heavy fluid of variable density. *Proc. Roy. Math. Soc.* **14**, 170–177.

Taylor, G. I. 1950 The instability of liquid surfaces when accelerated in a direction perpendicular to their plane. *Proc. Roy. Soc. London, Ser. A* **201**, 192–196.

Yokokawa, M., Itakura, K., Uno, A., Ishihara, T. & Yukio, K. 2002 16.4-Tflops direct numerical simulation of turbulence by a Fourier spectral method on the earth simulator. SC2002: Supercomputing Conference.