Cluster Computing in the SHMOD Framework on the NSF TeraGrid

Paul R. Woodward, Sarah E. Anderson, David H. Porter, and Anusha Iyer
paul@lcse.umn.edu, saraha@cray.com, dhp@lcse.umn.edu, anusha@cs.umn.edu

Laboratory for Computational Science & Engineering
University of Minnesota

I. Introduction.

Over the last several years, our team at the Laboratory for Computational Science & Engineering (LCSE) at the University of Minnesota has been developing cluster computing applications under our SHMOD framework (Shared Memory On Disk) in order to achieve the benefits of fault tolerant, dynamically load balanced computation with no compromise in delivered performance [13-16]. The SHMOD framework provides a natural extension of the concept of shared memory from a single machine to an entire cluster. It is based upon the concept of shared data objects that are implemented as disk files, even though these might actually reside in memory (in the disk buffer cache or on a RAM disk). In this respect it is based on ideas similar to the early Linda system and to efforts to build shared file systems (cf. [17-21]). The use of the disk file abstraction, however, separates the SHMOD approach from efforts to implement software distributed shared memory (cf. [1-4]), as well as from alternative out-of-core cluster computing approaches such as global arrays (cf. [5-12]).

An obvious limitation of the SHMOD approach is that applications must be restructured so that they require only, or essentially only, serial access to shared data. Also, the application should be structured to ingest shared data in large gulps rather than in small sips. A large granularity of the shared files is critical to overcoming the limitations of data access latency. Finally, reading and writing of shared data files should be overlapped completely with computation, so that it comes at essentially no performance cost. Restructuring of application codes to conform to this shared data access model can represent a significant cost; however, once the application is restructured, the benefits of SHMOD are numerous. In this paper, we outline the principal features of our present SHMOD implementation and illustrate their effectiveness through the example of a Mach 1 homogeneous turbulence simulation on a varying number of TeraGrid nodes at NCSA using a 2048^3 computational mesh. The SHMOD framework has proved to be highly scalable (up to 330 Gflop/s delivered performance on all available nodes, namely 250, at NCSA) and highly flexible (persistent computation through all kinds of events on dynamically adjusting numbers of nodes).

Our SHMOD approach is built upon a simple remote I/O server, the sRIO and ipRIO servers (www.lcse.umn.edu/shmod), that were built at the LCSE by Sarah Anderson and Anusha Iyer and that utilize standard file systems at nodes. Originally, we utilized MPI to send control messages between collaborating processes. However, MPI limited us to a constant number of network nodes, and it did not allow us to react easily to node failures. We therefore replaced the functionality that it was providing for our SHMOD applications with a globally accessible mySQL database. Other efforts needing to manage large numbers of files and/or tasks on a network have used similar database coordination mechanisms [22]. In general, our effort differs from data grid efforts based on Globus [23,24] in that we seek higher delivered data access bandwidths to support a tightly coupled computational model and we have so far restricted our attention to execution on single cluster systems. The importance of high delivered network bandwidth will become clear in the discussion below.

II. Structure of a SHMOD Application.

A SHMOD application must have a special structure. The application must be broken down into a series of separate tasks arranged in the order in which they should be launched. This is a recommended launch order, and it can be altered as a result of conditions that can only be known at run time. However, it is assumed that any task in this ordered list can be launched if a specific set of preceding tasks on the list have been completed. These task completion conditions for launch can be included as part of the task list, or they can be written into the task selection logic of the application itself. Each task operates upon a designated set of named data objects that we implement in the form of disk files. (If there is sufficient memory, these files will in fact be memory resident, because the operating system will automatically cache them.) We will refer to the set of disk files upon which a task operates as its "data context." The task can always be launched as soon as its data context is ready. This condition on the data context can always be translated into a condition on the completion of a set of previous tasks. We enforce the simplifying discipline that we do not dive below the granularity defined by the task decomposition in order to refine constraints upon events such as task launches. The output of each task is a new set of data files, which may in some cases be overwritten on earlier files that are no longer in use. It is an essential simplifying feature of our SHMOD applications that once a task is launched, it may run until completion without interruption and without any communication with any other task.

The very high parallel performance of SHMOD applications comes from the decomposition of the application into so large a number of tasks that at just about any point in the ordered task list many more tasks can be executing simultaneously than there are network nodes available to perform those tasks. So long as this condition is fulfilled, and it is usually possible to guarantee that it is for jobs of sufficient scale, all available network nodes can always be kept busy. If there is sufficient network bandwidth, then any participating network node that becomes available for work can simply grab the next task on the ordered list, since there need be no preference for operations upon local data. However, the network bandwidth requirements can be minimized by making intelligent task selections that take data location and deliverable bandwidth to the data into account along with position in the task ordering.

A. Structure of a SHMOD Task.

All tasks in a SHMOD application have the same basic structure. These tasks are performed by various server processes. Such a server is started up on a network node whenever that node becomes available for use by the application. On a remote super-

computer center, nodes are obtained through batch queue requests, and once the nodes are granted, the server processes are started up. On our local systems, we implement the server processes as system services and as screen savers. In either case, the first action of such a server is to look for work in a globally accessible mySQL database. In this model, the management function appears passive, since it appears to be implemented in a passive database (you call it, it does not call you). However, a management executive routine is incorporated in each server's code that allows the server to interpret the contents of the database and to make intelligent decisions about task selection based upon this global knowledge and a common set of logical rules.

This mode of operation permits simple fault tolerance, as we shall see, and it allows servers to come and go without disrupting the work flow. It does, however, introduce an issue of version control which has to be managed. This model also requires that upon task selection each server process must acquire the global knowledge of the database, think about it, modify it, and put it back. Clearly, this is critical code and the database locking mechanisms must be used when executing it. The key point that makes this approach practical is that the required global knowledge is very small, and the time to transmit it to the server is essentially the same as the time to contact the server at all (this database communication in SHMOD is limited by latency and not by bandwidth). Also, the server can think every possible useful thought about this global information in, say, a millisecond, so that this time interval is also negligible. Under these circumstances, there is no advantage to making a single intelligent manager process as opposed to a myriad of identical intelligent manager subroutines (distributed management).

Once the server has selected its next task, it reads the data context for that task from the locations specified in the task record in the database. These read operations could be from local memory, local disk, remote memory, or remote disk, but in each case they are logical disk file read operations that are specified by file names and paths, byte offsets into the files, and numbers of bytes to be read. In order to support this file read mechanism of data access, the application must be structured so that it takes in necessary data in a small number of large sequential read operations. This structure assures that performance will be bandwidth limited rather than latency limited. This restructuring is not hard to do once one realizes that this operation is just a data copy. We do not identify useful data and then access it repeatedly over the course of the computation. Instead, we pay the price of an optimized sequential copy at the outset (and another one at the end) which allows us to put the data into our own local memory workspace, from which we can begin to work with it at the granularity of cache lines rather than, for example, that of 100 MB data records. This data copy is overlapped with computation in high performance implementations (see below), so that the CPU is kept busy and the cost of this copy is minimized. Most of the tasks we are talking about here have a truly enormous granularity, so that hundreds of MB are being copied at a time. Nevertheless, this copy of task data into a private workspace proves to be an excellent strategy at all levels of the memory hierarchy, extending upward to a CPU task within an SMP, in which case the shared data is brought onto the CPU's own board, and even to a subtask within that CPU task, in which case the data is brought into the cache memory of the processor before being operated upon. Thus, even though our strategy of copying data may at first seem wasteful, it is a natural extension of best practices at all levels of today's memory hierarchy.

Once the task data context has been copied into the network node's local memory, the server operates upon it according to the particular application, be it gas dynamics, scientific visualization, or any other application. The organization of the data in the local memory into relatively small cache lines means that this data can now be effectively transposed, for example, or accessed at a small level of granularity to deal with such application features as AMR grids or multimaterial flow. Once the server has finished operating on the data, it copies the result back to the globally shared data space by means of a small set of file write operations, with file paths, byte offsets, and byte lengths. Only when these data write-back operations are successfully completed does the server access the global database once more to register its task completion and to select a new task to perform.

B. Structure of a High-Performance SHMOD Task.

In order to obtain high performance from a SHMOD application, on most networks it is necessary to overlap the two data communication steps – the initial data reads and the final data writes – with computation. This is easily done by instantiating a separate communications thread for the task server. The global control database is still accessed only once per task, after all writes are successfully completed, but now this occurs during the execution of a subsequent task. Upon this database access a new task is selected. The data context for that task is read while the present task computation continues, so that once that computation is complete the server will be able to proceed immediately to the computation for the next task. This requires holding one additional copy of a task data context at the server's node throughout the computation, but it has so far been easy to find a task granularity that makes this possible while still delivering excellent local and excellent parallel performance.

It is important to note that the desire to hide all data transfer costs behind overlapped CPU computation tends to set limits on the task granularity. We have already seen that this requirement forces tasks to be smaller than they might be, so that an extra task data context can fit into the local memory. Since we usually need at least 2 full copies (an old and a new) of the data context for the task computation itself, the need to fit one more into the local memory reduces the maximum task granularity by only a third. However, the need to overlap communication with computation forces the task granularity up, since for most tasks we do there is more useful computation that can be performed on each word of the data the larger the task data context becomes. Quite generally, larger tasks can be usefully made more computationally intensive, and this gives us more time to transmit the necessary data. How these issues play out in practice is illustrated in the example discussed below.

III. Characteristics of the PPM Numerical Method that Permit a Practical SHMOD Implementation.

The PPM gas dynamics codes [25-28] have been in use for decades to simulate compressible flows involving strong shocks and multimaterial boundaries. These codes have enjoyed very high performance parallel implementations on a number of very different computer architectures (cf. [13]). A reason that this has been possible, aside from a willingness of our team to rewrite these codes again and again, is that this numerical method has 3 very special properties:

1. The numerical method is *explicit*, which means that it updates the fluid state in a grid cell using only information contained in that cell and in its near neighbors within a particular *difference stencil*.
2. The numerical method employs *directional operator splitting*, which means that it is formulated in terms of a sequence of 1-D

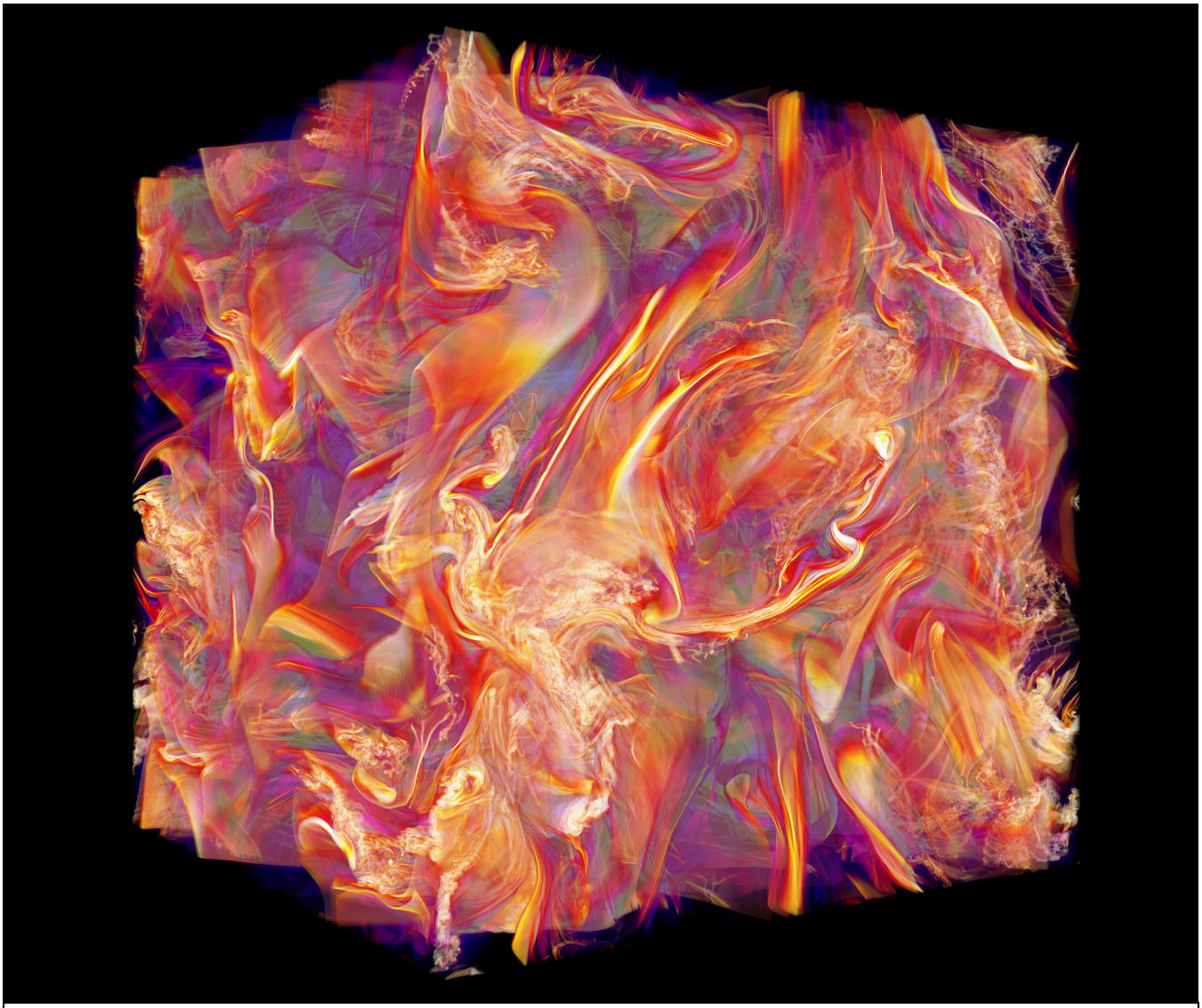


Fig. 1A. Volume visualization using the LCSE's utility HVR (www.lcse.umn.edu/hvr) of the distribution of the magnitude of the vorticity in the PPM simulation of Mach 1 homogeneous turbulence discussed in the text. The flow in the cubic domain, with periodic boundary conditions, is shown as some regions near strong shear layers begin to become turbulent. Blue regions have the weakest vorticity values, and as the vorticity increases in strength, the color goes through red to yellow and finally to white.

passes, each of which treats derivatives of the flow quantities only in a single grid direction.

3. The numerical method is of *high accuracy*, which means that it performs a relatively large amount of computation on each data element in order to achieve a good representation of the gas dynamics.

It is easiest to see how these features permit a SHMOD implementation in the light of a specific example.

IV. The Turbulence Problem.

Nearly everyone has at one time or another been advised to fasten his or her seatbelt because of an encounter with unanticipated turbulence. In astrophysics, turbulence is nearly always anticipated, since the miniscule viscosity of astrophysical flows and their nearly universal instability gives rise to turbulence almost everywhere. For this reason, simulation of astrophysical flows can be improved by embedding models of small-scale turbulence into the governing equations, or equivalently into the numerical simulation algorithms. For many years our team at the LCSE has been simulating homogeneous, compressible turbulence as well as turbulent convection in stars in the hope of producing better subgrid-scale models of the turbulence phenomenon for use in astrophysical simulations [29-37].

In September and October of 2003, our team ran a PPM simulation of Mach 1 homogeneous turbulence on a grid of 2048^3 cells on the NCSA TeraGrid cluster of 250 dual-processor 1.3 GHz Itanium-2 nodes. We used a SHMOD implementation to keep this run going in a friendly user period while the availability of nodes was varying quite widely over time. All of the data, some 5 TB, from this simulation was sent to the San Diego Supercomputer Center using the Storage Resource Broker, and we sent a copy of 2 TB of this data – a time history of the evolution of the vorticity of the gas – to our lab in Minnesota for the purpose of generating visualizations of the flow dynamics. A few of these visualizations appear at the right on these pages, and movies made from this data can be found on our Web site at

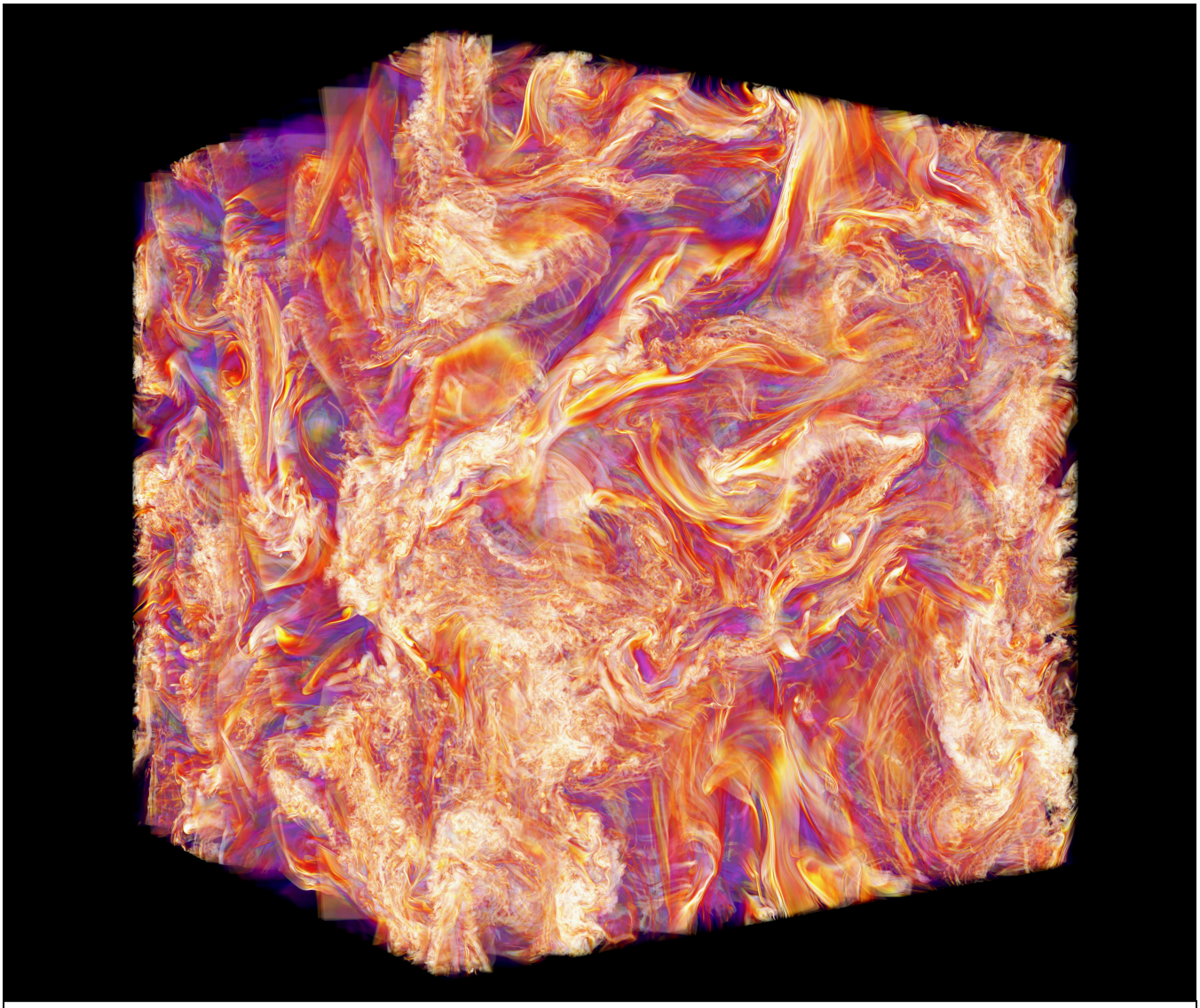


Fig. 1B. Volume visualization of the distribution of the magnitude of the vorticity in the PPM simulation of Mach 1 homogeneous turbulence discussed in the text. The flow is shown about midway through the transition from the original, smoothly stirred flow to fully developed turbulence. Patches where the turbulent transition is complete can be seen as well as regions containing shear layers that are beginning to become unstable. The relation between color and vorticity magnitude is similar to that in Fig 1a.

www.lcse.umn.edu/AVIs. Initial analysis of some of the more quantitative data reveal that the behavior of the gas in this simulation is consistent with our team's theoretical ideas about small-scale turbulence (cf. [34-36]).

A. Task Granularity.

For the purpose of this run, we decomposed our 8.6 billion cell grid into 1024 grid bricks, each with 128×256^2 cells. In order to update each of these bricks for two full time steps as an independent task, we needed to augment each brick with face, edge, and corner chunks that came either from the boundary conditions (periodic in each dimension) or from portions of neighboring bricks. These "ghost cell" chunks were 20 grid cells thick. This surface-to-volume ratio, together with the fact that we can progressively shrink the ghost cell regions as we perform sweep after sweep of the PPM algorithm, meant that ¼ of the computational labor performed in each brick update was redundant with work performed in the updates of neighboring bricks. This cost of redundant computation, which is introduced to increase the task granularity, is the principal cost overhead of our SHMOD implementation. Note that it is a constant fraction of the labor regardless of the number of participating nodes or of the number of grid bricks in the problem.

The task granularity we chose for this problem resulted in 1024 tasks per round of grid brick updates. Since we had at most 250 nodes available to us, this meant that there were always, on average, 4 or more tasks for each node to perform in each round of updates. This fact resulted in the ability of nodes to proceed to the next round of updates before all bricks of the previous round were updated. Consequently, all nodes were constantly kept busy. However, this also meant that at the beginning of each round of updates (that is, at the beginning of a pair of time steps) we did not have all the data from the previous round to bring to bear on the task of choosing the value for the next time step interval. As a result we chose this time step value speculatively. We used all available data, and in the event that our choice proved too large,

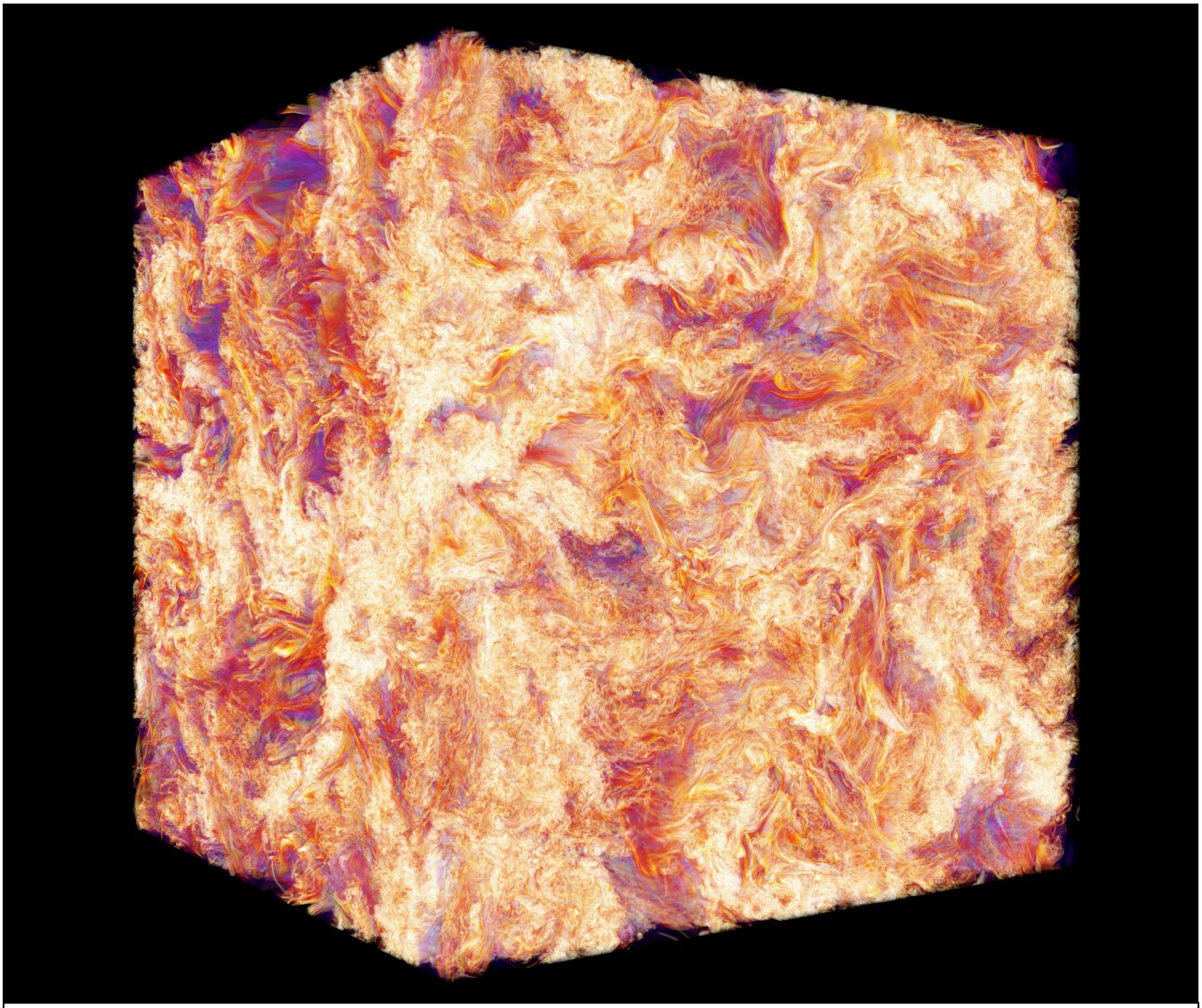


Fig. 1C. Volume visualization of the distribution of the magnitude of the vorticity in the PPM simulation of Mach 1 homogeneous turbulence discussed in the text. The transition from the original, smoothly stirred flow to fully developed turbulence is now well underway. Few shear layers remain that have not developed into large tangles of intertwined vortex tubes. Nevertheless, patches (blue regions) of relative calm remain in this turbulent flow. The relation between color and vorticity magnitude is similar to that in Fig 1a.

which essentially never happened in this particular problem, we were prepared to roll back the computation and to begin that round of updates all over again. The time step value is the result of a global reduction operation. We used data from previous time steps to produce a speculative value. We were conservative, which produces a guess that is systematically a bit smaller than it needs to be, but that avoids producing the over-large value that might force us to roll the computation back. By doing this we paid a very small price in taking a slightly larger number of time steps than would in principle be necessary, but we saved the time that many nodes would otherwise have been forced to waste waiting upon stragglers to complete each time step update round. This technique represents a general approach that one can adopt to the handling of global reductions in SHMOD applications.

Our chosen task granularity had other consequences. We obtained 660 Mflop/s sustained computational performance from each participating CPU, averaged over the entire computation and over all CPUs participating. This is 13% of the Itanium-2 peak performance, which seems to be a fairly good delivered performance level for this CPU. It was achieved in part because the entire workspace for the 1-D update of a single grid strip and for the preparation of that data fits into the CPU's 3 MB cache memory. PPM performs about 960 flops to update each grid cell for a single 1-D sweep, and there are 6 such sweeps (over a grid brick of progressively shrinking size) during one grid brick update task. For our grid bricks of 128×256^2 cells, the amount of work this involves is about 6 times the number of flops that would be required to update the average size of the brick during this process, namely 148×276^2 cells, for a single 1-D sweep. This is 10.8 Gflops, and at the 1.32 Gflop/s aggregate performance of the node's dual CPUs this takes about 49 seconds. Now the reason for our choice to update each grid brick for 2 time steps in a single task rather than just for 1 time step should be clear. This gives us almost twice the time for the data from the previous update to be written back and for the data for the next update to be acquired over the network. It also roughly halves the frequency of node

Fig. 1D. Volume visualization of the distribution of the magnitude of the vorticity in a thin slice of the PPM simulation of Mach 1 homogeneous turbulence discussed in the text. The relation between color and vorticity magnitude is identical to that in Fig 1b. Here the flow is shown a the time of Fig 1b, viewed closer up but from the same angle and with clipping planes making only a slice of the domain near the center visible. The slice shown extends across the entire diagonal width of the cubical problem domain, revealing the wealth of detail that is captured by the PPM scheme operating on this very fine, 2048^3 grid. This high degree of captured detail allows predictions of alternative turbulence closure models to be compared with the simulation data to generate statistically significant quantitative measures of their predictive capability.

requests to lock, acquire, modify, and replace the contents of our small global database. There are, with all 250 nodes cooperating, an average of only 5 such database hits per second.

In designing a SHMOD application, it is important to understand the relationship between task granularity, parallel computation overhead (redundant computation in ghost cell regions), and network bandwidth requirements. (These are embedded in a PPM parallel performance spreadsheet described and available at www.lcse.umn.edu/PPM-performance-spreadsheet.) The larger we make our tasks, the longer we generally will have to communicate the necessary data. The proper choice of granularity represents a balance between the needs to have many more parallel tasks than processing nodes, to have the tasks fit into the node memories, to have enough time to send the data over the network while the CPU is busy with computation, and to have a sufficiently low frequency of interaction with the global database. The acceptable ranges for each of these considerations is sufficiently broad that we have so far had no difficulty in satisfying all these constraints simultaneously on every system on which we have run. We believe that the turbulence computation reported here could easily have scaled up to 1024 nodes, had they been available, without significant loss in per-node performance.

Perhaps the most unusual aspect of SHMOD computation is the enormous amount of data movement over the network that it involves. For this reason it is difficult to appreciate that only modest networking requirements flow from this feature of our paradigm. The reason for this is that all network communications can be implemented as asynchronous background tasks, so that the network is in continuous, rather than sporadic use. Applications that have been restructured for SHMOD implementation do not have any significant network latency requirements. For SHMOD, it is only network bandwidth that usually matters, and that is almost always more than sufficient on modern networks, at least for PPM computation. For example, in the case reported here, if each node needed to communicate every grid brick over the network, taking 49 seconds to read one in and to write one back, then the necessary network bandwidth to each node would be only 9.0 MB/sec (equivalent to Fast Ethernet). This estimate assumes that the node data all comes from other nodes on the network, and that the node at issue never needs to serve any data that it owns to other nodes. If all the nodes serve each other data, then we would need about twice this network bandwidth, or only 18 MB/sec. Since our data transmissions were sent on the Myrinet interconnect of the NCSA TeraGrid cluster at 100 to 150 MB/sec, sustained, we obviously had plenty of bandwidth on this

system. For files stored on the single disks at each node and not in the node's disk buffer cache, there was also enough bandwidth to handle this requirement. It was therefore unnecessary for us to implement any favoritism for tasks that involve data on a given node, but which might not be at precisely the best position in the ordered task list.

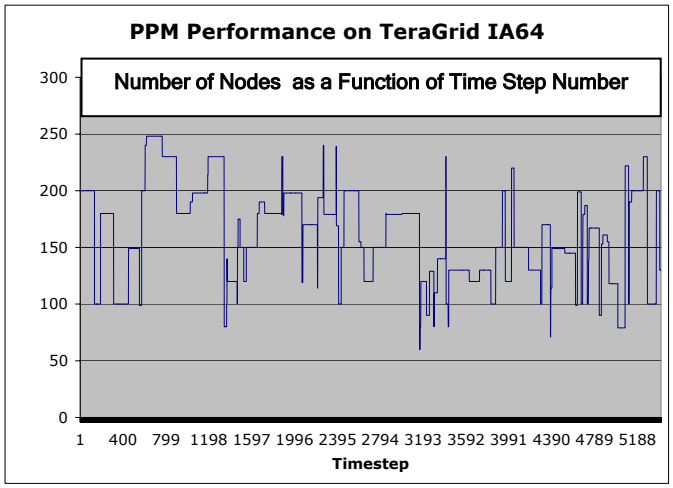
Note that we were able to make the choices of task granularity reported above because the job we set out to do was very large. It would have been much harder to use the same large number of nodes efficiently to do a much smaller problem. In fact, if we were to scale our problem down to a size so small that it could be completed in only, say, a single hour, then SHMOD computation would appear much less attractive. This is not a problem, because small problems, in our experience, consume only a minor fraction of the CPU time that we use. Small problems can be done efficiently in the SHMOD paradigm on small resources. For example, at the same time that this simulation ran, we were running a much smaller, one billion cell PPM simulation of shear layer instability at the same high efficiency level under the SHMOD paradigm on our own local 32-CPU Unisys ES-7000. Very small problems tend to have different requirements, such as ease of setup and immediate presentation of graphical results, so that efficiency is not the primary concern for these runs. It is a fundamental and very important aspect of the SHMOD approach that it looks increasingly more attractive at increasingly large scale. It is important to realize that a computational technique that is very well suited to very large scale computation may not be the paradigm of choice for small runs. And conversely, methods that work well for modest problems may barely work at all when attempted at scale.

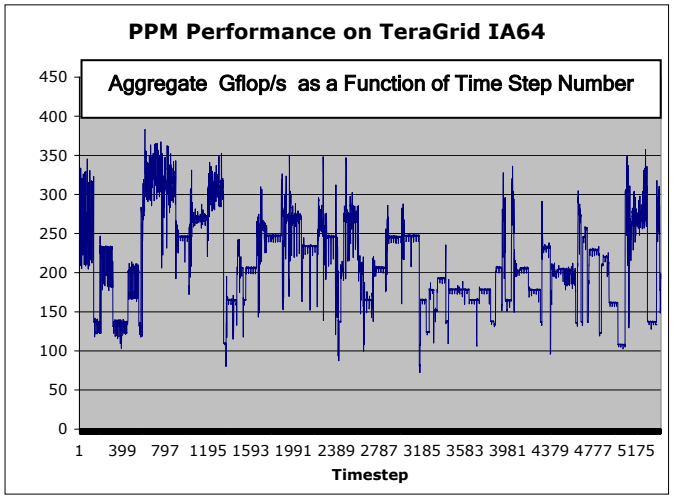
B. Flexible, Fault Tolerant Job Execution.

One of the greatest benefits of SHMOD is an extremely flexible and fault tolerant job execution model. The reason for this is the persistent problem image implemented in the many separate task data contexts, which are files resident on nonvolatile disk storage (these are dumped to disk when the run is interrupted, even if during the run they are cached in memory by the operating system). If we absorb a bit more of our network bandwidth, we can even mirror these data contexts on multiple network nodes, so that the loss of a node will produce barely a glitch in the ongoing computation. The second feature of SMOD computation that is responsible for highly flexible job execution is the independence of the computing and the storage resources. A new node can join the computation at any point, select a task from the global database, and perform that task regardless of the location of its data context. With equal ease, nodes can leave the ongoing job in order to do work for other users. (See the performance logs in Fig. 2.) This means that enormous fluid dynamical simulations can actually run as background jobs at supercomputer centers. With equal force we may conclude that SHMOD allows fluid dynamicists with access to only modest resources to compute the largest problems, if only they are willing to wait long enough. In fact, SHMOD elevates the disk from the level of storage to the level of memory in the functional hierarchy. In so doing, SHMOD has the potential to very significantly reduce the cost of high performance fluid dynamical computation by replacing in systems designed for this purpose most of the RAM, usually the costliest system component, with disk, which is usually the cheapest component.

C. Working Data.

A diagram of the system we used is shown in Fig. 3. The performance logs for this job shown in Fig. 2 indicate that it was





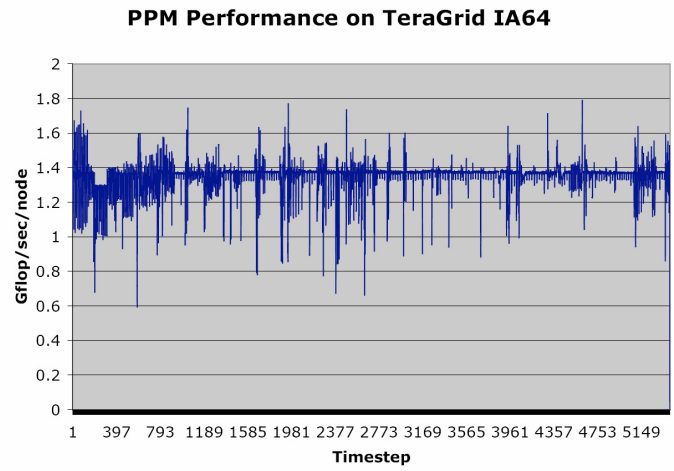


Fig. 2. Scalability the 2048^3 PPM turbulence run.

always possible to find 80 nodes available on this system during the friendly user period when we were running. A single active problem image consisted of 1024 files of 160 MB each, so that with 80 nodes acting as file servers for the entire problem (these nodes also participated in the computation), one such problem image could be accommodated using half the 4 GB memory on each of these nodes. We always kept both an old and a new

problem image, so that not all of this data could be memory resident on the 80 nodes so long as we also used them for computation. Nevertheless, the single 60 GB scratch disk on each node was able to keep up. The performance logs clearly show that the per-node performance was essentially independent of the number of nodes participating. This means that not only were the 80 nodes able to serve data contexts to as many as 250 total nodes, but also that this service had a negligible impact on their computational performance. It is also important to realize that since the memories of about 160 nodes were required to hold all the working data for this computation, during close to half of this run we could not have executed our code at all on this system without the benefit of SHMOD memory on disk.

V. Future Work.

We have also used our SHMOD framework to implement our utility programs that post-process and visualize our simulation data on multiple PC clusters [38]. We would like to integrate the operation of these multiple stages of the fluid flow simulation and visualization pipeline, now automated individually for unattended execution, into a single, fully automated workflow. Once we set a run going we would like to obtain data analysis and visualization products automatically as it executes, and we would like to be able to interact with the run on demand to change these default products or to change run parameters such as the frequency of generation of these products. We would also like to integrate this software with automated transport of files over wide area networks using the Globus toolkit, in order to achieve a fully automated, continental-scale Grid-based implementation of the workflow.

VI. Acknowledgements:

We would like to thank the TeraGrid team and especially Rob Pennington's team at NCSA for their cooperation on this project, which was carried out under a friendly user period. This work built on earlier experiments in SHMOD computing at NCSA supported by the NSF PACI program through NCSA (grant ACI-9619019, subaward 786) and aided by considerable assistance from Rob Pennington's team. The development of the PPM algorithm and its associated data analysis and visualization software has been supported by DoE through the Office of Science (DE-FG02-03ER25569) and the ASCI program as well as by NSF through the PACI program and a CISE Research Resources grant (CNS-0224424). We have also enjoyed generous local support through the University of Minnesota's Minnesota Supercomputer Institute and Digital Technology Center. We also acknowledge help from the Intel compiler team in our first optimizations of PPM for the Itanium processor family and a donation by Unisys of a 32-processor ES-7000 on which we developed and optimized the more recent Itanium-2 versions of our codes under the SHMOD paradigm.

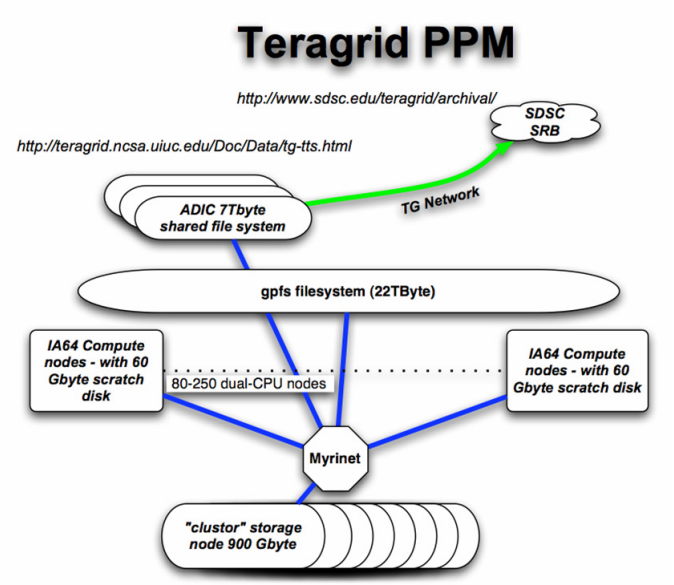


Fig. 3. System diagram for the PPM turbulence run.

VII. References:

1. Amza, C., Cox, A. L., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W., and Zwaenepoel, W., 1996. "TreadMarks: Shared Memory Computing on Networks of Workstations," *IEEE Computer*, Vol. **29**, No. 2, pp. 18-28, Feb. 1996. (available at http://www.cs.rice.edu/~willy/TreadMarks/papers.html)
2. Scales, D., and Lam, M., 1994. "The Design and Evaluation of a Shared Object System for Distributed Memory Machines," Proc. First Symposium on Operating Systems Design and Implementation, Nov. 1994.
3. Erlichson, A., Nuckolls, N., Chesson, G., and Hennessy, J., 1996. "SoftFLASH: Analyzing the Performance of Clustered Distributed Virtual Shared Memory," Proc. of the 7th Int. Conf. On Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, pp.210-20. (available at http://wwwflash.stanford.edu/architecture/papers/paperlinks.html)
4. Bilas, A., Iftode, L., and Singh, J. P., 1998. "Evaluation of Hardware Support for Shared Virtual Memory Clusters," Proc. of the 12th ACM International Conf. on Supercomputing (ICS'98), Melbourne, Australia. July, 1998.
5. Nieplocha, J., Harrison, R., and Littlefield, R., 1994. "Global Arrays: A Portable 'Shared-Memory' Programming Model for Distributed Memory Computers," Proc. Supercomputing '94, pp 340-349; available at www.emsl.pnl.gov/docs/global/papers/super94.pdf.
6. Nieplocha, J., Harrison, R., and Ian Foster, 1996. "Explicit Management of Memory Hierarchy," *Advances in High Performance Computing*, Ed. L. Grandinetti, J. Kowalik, M. Vajtersic, NATO ASI 3/30: pp 185-198; available at www.emsl.pnl.gov/docs/global/papers/nato.pdf
7. Nieplocha, J., and Foster, I., 1996. "Disk Resident Arrays: An Array-Oriented I/O Library for Out-of-Core Computations," Proc. IEEE Conference on Frontiers of Massively Parallel Computing, Frontiers '96, pp 196-204.
8. Nieplocha, J., Harrison, R. J., Kumar, M. K., Palmer, B., Tipparaju, V., and Trease, H., "Combining Distributed and Shared Memory Models: Approach and Evolution of the Global Arrays Toolkit," Proc. POOHL 2002 workshop of ICS-2002, NYC, 2002; available at www.emsl.pnl.gov/docs/global/papers/jarek.pdf.
9. Palmer, B., and Nieplocha, J., "Efficient Algorithms for Ghost Cell Updates on Two Classes of MPP Architectures," Proc. PDCS-2002; available at www.emsl.pnl.gov/docs/global/papers/ghosts.pdf.
10. Salmon, J., and Warren, M. S., 1997. "Parallel Out-of-Core Methods for N-Body Simulation," Proc. 8th SIAM Conf. On Parallel Processing for Scientific Computing.
11. Fink, S. J., and Baden, S. B., 1997. "Runtime Support for Multi-Tier Programming of Block-Structured Applications on SMP Clusters," *Lecture Notes in Computer Science*, ed. Y. Ishikawa et al., Vol. **1343**, 1-8. (available at http://www-cse.ucsd.edu/groups/hpcl/scg/kelp.html)
12. Baden, S. B., and Fink, S. J., 1999. "A Programming Methodology for Dual-Tier Multicomputers," *IEEE Transactions on Software Engineering*, **26**, 212-26 (2000). (available at http://www-cse.ucsd.edu/groups/hpcl/scg/kelp.html)

13. Woodward, P. R., 1996. "Perspectives on Supercomputing: Three Decades of Change," *IEEE Computer*, Vol. **29**, Oct. 1996, pp. 99-111. An edited manuscript of this paper, with illustrations, is available at http://www.lcse.umn.edu/computer.
14. Woodward, P. R., and S. E. Anderson, "Portable Petaflop/s Programming: Applying Distributed Computing Methodology to the Grid Within a Single Machine Room," Proc. of the 8th IEEE International Conference on High Performance Distributed Computing, Redondo Beach, Calif., Aug., 1999; available at www.lcse.umn.edu/HPDC8.
15. Anderson, S. E., B. K. Edgar, D. H. Porter, and P. R. Woodward, "Cluster Programming with Shared Memory on Disk", Proceedings of "Linux Clusters: the HPC Revolution" conference, June 25-27, 2001, Urbana, IL; available at http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF01/Anderson_uminn.pdf
16. Anderson, S. E., 2003, SHMOD library. www.lcse.umn.edu/shmod.
17. Soltis, S., T. Ruwart, and M. O'Keefe, "The Global File System," Proc. 5th NASA Conference on Mass Storage Systems and Technologies, Sept., 1996.
18. Soltis, S., G. Erickson, K. Preslan, M. O'Keefe, and T. Ruwart, "The Design and Performance of a Shared Disk File System for IRIX," Proc. 15th IEEE Symposium on Mass Storage Systems, March, 1998.
19. Elder, A., T. M. Ruwart, B. D. Allen, A. Bartow, S. E. Anderson, and D. H. Porter, 2000, "The InTENsity PowerWall: A Case Study for a Shared File System Testing Framework," Proc. 17th IEEE Symposium on Mass Storage Systems / 8th NASA Goddard Conference on Mass Storage Systems and Technologies, March, 2000.
20. Cluster File Systems, Inc., 2002. "Lustre: A Scalable, High-Performance File System," available at www.lustre.org/docs/-whitepaper.pdf
21. Panasas, Inc., 2003. "Panasas ActiveScale File System," available at www.panasas.com/docs/Panasas_ActiveScale_DS.pdf
22. NCSA MEAD Expedition (Modeling Environment for Atmospheric Discovery) home page, www.ncsa.uiuc.edu/expeditions/MEAD.
23. Foster, I., D. Kohr, R. Krishnaiyer, and J. Mogill, "Remote I/O: Fast Access to Distant Storage," Proc. Workshop on I/O in Parallel and Distributed Systems (IOPADS), pp. 14-25, 1997.
24. Allcock, B., J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke, "Data Management and Transfer in High Performance Computational Grid Environments," *Parallel Computing Journal*, **28**, 749-771 (2002).
25. Woodward, P. R., and Colella, P., 1984. "The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks," *J. Comput. Phys*. **54**, 115-173.
26. Colella, P., and Woodward, P. R., 1984. "The Piecewise-Parabolic Method (PPM) for Gas Dynamical Simulations," *J. Comput. Phys*. **54**, 174-201.
27. Woodward, P. R., 1986. "Numerical Methods for Astrophysicists," in *Astrophysical Radiation Hydrodynamics*, eds. K.-H.Winkler and M. L. Norman, Reidel, 1986, pp. 245-326.
28. Edgar, B. K., Woodward, P. R., Anderson, S. E., Porter, D. H., and Dai, Wenlong, 1999. PPMLIB home page at http://www.lcse.-umn.edu/PPMlib.
29. Woodward, P. R., 1994. "Superfine Grids for Turbulent Flows," *IEEE Computational Science & Engineering*, Vol. **1**, No. 4, pp. 4-5+cover illustration, (December, 1994).
30. Porter, D. H., and Woodward, P. R., "3-D Simulations of Turbulent Compressible Convection," *Astrophysical Journal Suppl.* **127**, 159-187 (2000); available at www.lcse.umn.edu/conv3d.
31. Porter, D. H., Woodward, P. R., and Jacobs, M. L., "Convection in Slab and Spheroidal Geometries," Proc. 14th International Florida Workshop in Nonlinear Astronomy and Physics: Astrophysical Turbulence and Convection, Univ. of Florida, Feb., 1999; *Annals of the New York Academy of Sciences Vol. 898, pp. 1-20,* (2000); available at www.lcse.umn.edu/convsph.
32. Sytine, I. V., D. H. Porter, P. R. Woodward, S. H. Hodson, and K.-H. Winkler, "Convergence Tests for Piecewise-Parabolic Method and Navier-Stokes Solutions for Homogeneous Compressible Turbulence," *J. Computational Physics* **158**, 225-238 (2000).
33. Green, K., 2001. "Going with the Flow," NCSA Access Magazine, Fall 2001, available at www.ncsa.uiuc.edu/News/Access/-Stories/itaniumflow.
34. Woodward, P. R., D. H. Porter, I. Sytine, S. E. Anderson, A. A. Mirin, B. C. Curtis, R. H. Cohen, W. P. Dannevik, A. M. Dimits, D. E. Eliason, K.-H. Winkler, and S. W. Hodson, 2001, "Very High Resolution Simulations of Compressible, Turbulent Flows," in *Computational Fluid Dynamics,* Proc. of the 4th UNAM Supercomputing Conference, Mexico City, June, 2000, edited by E. Ramos, G. Cisneros, R. Fernández-Flores, A. Santillan-González, World Scientific (2001); available at www.lcse.umn.edu/mexico.
35. Cohen, R. H., W. P. Dannevik, A. M. Dimits, D. E. Eliason, A. A. Mirin, Y. Zhou, D. H. Porter, and P. R. Woodward, "Three-Dimensional Simulation of a Richtmyer-Meshkov Instability with a Two-Scale Initial Perturbation," *Physics of Fluids*, **14**, 3692-3709 (2002).
36. Woodward, P. R., Porter, D. H., and Jacobs, M., "3-D Simulations of Turbulent, Compressible Stellar Convection," in *3D Stellar Evolution*, S. Turcotte, S. C. Keller, and R. M. Cavallo, eds., *ASP Conf. Series*, Vol. **293**, 45-63 (2003); available at www.lcse.umn.edu/3Dstars.
37. LCSE movie animations of the run reported here and similar SHMOD computations can be viewed at www.lcse.umn.edu/AVIs.
38. Porter, D. H., Woodward, P. R., and Iyer, A. 2004. "Initial Experiences with Grid-Based Volume Visualization of Fluid Flow Simulations on PC Clusters," submitted to IEEE VolVis 2004 conference; available at www.lcse.umn.edu/hvr.