Harnessing the Power of the New SMP Cluster Architecture

P.R. Woodward, S.E. Anderson, D.H. Porter, D. Dinge, I. Sytine,
T. Ruwart, M. Jacobs, R.H. Cohen, B.C. Curtis, W.P. Dannevik,
A.M. Dimits, D.E. Eliason, A.A. Mirin, K-H. Winkler and S. Hodson

Lawrence
Livermore
National
Laboratory

Center for
Applied Scientific Computing

UCRL-JC-134547
June 1999

This work was performed under the auspices of the U.S. Dept. of Energy at LLNL under contract no. W-7405-Eng-48.

Harnessing the Power of the New SMP Cluster Architecture

Paul R. Woodward, Steven E. Anderson, David H. Porter, Dennis Dinge, Igor Sytine, Thomas Ruwart, Michael Jacobs
Laboratory of Computational Science & Engineering, University of Minnesota
and
R. H. Cohen, B. C. Curtis, W. P. Dannevik, A. M. Dimits, D. E. Eliason, A. A. Mirin
Lawrence Livermore National Laboratory
and
Karl-Heinz Winkler and Stephen Hodson
Los Alamos National Laboratory

Introduction:

In 1993, members of our team collaborated with Silicon Graphics to perform the first full-scale demonstration of the computational power of the SMP cluster supercomputer architecture [1]. That demonstration involved the simulation of homogeneous, compressible turbulence on a uniform grid of a billion cells, using our PPM gas dynamics code [2-5]. This computation was "embarrassingly parallel," the ideal test case, and it achieved only 4.9 Gflop/s performance, slightly over half that achievable by this application on the most expensive supercomputers of that day. After four to five solid days of computation, when the prototype machine had to be dismantled, the simulation was only about 20% completed. Nevertheless, this computation gave us important new insights into compressible turbulence [6,7] and also into a powerful new mode of cost-effective, commercially sustainable supercomputing [8]. In the intervening 6 years, the SMP cluster architecture has become a fundamental strategy for several large supercomputer centers in the U.S., including the DoE's ASCI centers at Los Alamos National Laboratory and at the Lawrence Livermore National Laboratory and the NSF's center NCSA at the University of Illinois. This SMP cluster architecture now underlies product offerings at the high-end of performance from SGI, IBM, and HP, among others. Nevertheless, despite many successes, it is our opinion that the computational science community is only now beginning to exploit the full promise of these new computing platforms. In this paper, we will briefly discuss two key architectural issues, vector computing and the flat multiprocessor architecture, which continue to drive spirited discussions among computational scientists, and then we will describe the hierarchical shared memory programming paradigm that we feel is best suited to the creative use of SMP cluster systems. Finally, we will give examples of recent large-scale simulations carried out by our team on these kinds of systems and point toward the still more challenging work which we foresee in the near future.

Vector or Scalar Computation on a Single CPU?

Computational scientists and numerical algorithm specialists still argue over the benefits of vector versus scalar computation on a single CPU. These disagreements are reflected in hardware offerings as well, but in the U.S., mainstream microprocessor manufacturers are designing CPUs principally with the goal of rapid execution of scalar code. These CPUs acknowledge certain mainstream commercial applications, such as signal processing and compression/decompression of multimedia data streams as well as interactive 3-D graphics for CAD applications, Hollywood movies, and, more importantly, computer games, that vectorize in a natural way. However, these specialized vector applications do not require high floating point precision. These processors therefore support short, low-precision vector operations, but the usefulness of this hardware functionality for scientific computing applications familiar from the heyday of vector supercomputers in the U.S. (in the 1980's) is unclear.

In the realm of scientific computation, the vector versus scalar argument hinges on the fact that vector code, if a useful algorithm can be formulated in this way, relatively easily achieves half the peak potential CPU performance. For specialized linear algebra applications, vector code can come close to the peak potential of the CPU. The problem is that not every useful algorithm can be formulated in terms of code that is overwhelmingly of a vector nature. Once scatter-gather instructions enter the mix, performance tends to plummet (although this performance degradation is algorithm dependent). Additionally, the most favored applications, the dot products involved in linear algebra, perform at peak rates on CPUs of either type, whether vector or scalar. In the U.S., this vector versus scalar issue is being driven by powerful commercial forces beyond the control of even the wealthiest scientific communities. Apparently, outside of science there are not very many applications written in vector format (save of course the signal processing and graphics applications already noted). This situation is ameliorated by the fact that modern RISC CPU design is fundamentally based upon pipelining, the core technology behind vector computing.

The pipelined RISC CPU architecture assures that, so long as a scalar code presents to the CPU a sufficient frequency of executable flops within its instruction and data streams, performance should come close to that of vector code. Achieving near vector performance on scalar code apparently requires lots of registers, out of order instruction execution, speculative execution after branches, and the like – all standard tricks of today's highly sophisticated microprocessors. Assuming that the CPU has enough registers to keep temporary results within a moment's access range, and assuming that it also has a large, fast cache memory to draw upon, multiple iterations of a loop body, or even more general multiple threads, can be executed simultaneously in order to present within the combined instruction stream enough executable flops to keep functional units busy. In the 1970's, when vector computing was first introduced, vector performance was to be compared to that of relatively unsophisticated scalar processors without the benefit of cache memories (unless we consider the small core memory of the CDC 7600 as a cache). With such unsophisticated competition, vectors were a clear win. Today, however, the competition makes for a much closer call. The comparison is also algorithm dependent, as the older among us will remember that it was in the 1970's. The reason for this algorithm dependence is the fact that in order to force computation into a vectorizable mode it is usually necessary to perform unnecessary arithmetic. For the PPM gas dynamics codes we use, vectorization requires between 50% to 100% additional arithmetic, even for an ideal, embarrassingly parallel fluid dynamics problem [8,9].

Over the last few years, the scientific computing community has become familiar with the program structure requirements of cache-based microprocessors. In general, one needs to "block" numerical algorithms, so that a great deal of work is performed on a contiguous subset of the overall data while it resides in the cache memory. This is not the natural structure of a vector program, since it tends to result in relatively short vector lengths. However, it is the means of reducing demands on the bandwidth of the memory that is shared among the multiple processors in any of today's powerful computing systems. As more and more scientific computing applications are converted to such blocked numerical algorithms, it becomes harder and harder to find compelling examples for the preservation of classic vector computing. This conversion is not now complete, but it is progressing rapidly, at least in the United States. A conversion that is less rapid is one to truly scalar algorithms that no longer attempt to find compromise numerical treatments to make, for example, grid cells with shocks look mathematically like grid cells without them, or grid cells with multiple fluids look in some sense like single-fluid cells. Once one decides that it is no longer necessary to maintain a code that will run well on vector computers, the far more potentially accurate and powerful numerical algorithms that are truly scalar become available for consideration. This potential trove of powerful new algorithms is becoming practical just as supercomputing systems are attaining true, sustained teraflop/s computational capability, permitting the far more complex physical simulations that can best make use of the new algorithms.

Flat MPP Architecture or Cluster of SMPs?

A second architectural issue capable of driving earnest and extended arguments within the scientific computing community today is the one of CPU equality or CPU hierarchy. The first multiprocessor machines had so few CPUs that it was pointless to arrange them hierarchically. In the 2-, 4-, 8-, and 16-processor vector machines of the 1980's and early 1990's in the U.S., every processor was equal. This architecture of CPU equality was extended in a wide variety of machines known as massively parallel processors, or MPPs. Well-known examples from the U.S. industry are the Connection Machines, the Intel Paragons, the Cray T3Ds and T3Es, and the IBM SP-1. Even though the programmer might lay out these processors meaningfully in his or her program to form a 2-D or 3-D torus, for example, the advertised concept was that the processors were capable of all being treated equally. None had preferred access to any memory other than its own local one; or at least it was supposed to be possible to write an efficient program assuming that this was so. This was really a very appealing concept. However, its appeal tends to fade as the number of processors increases into the hundreds or thousands and, particularly, as one begins to attack dynamically irregular scientific computing problems on the machine. Dynamic load balancing on such MPP platforms with over 500 processors is a rare feat, requiring a programming tour de force.

The SMP cluster architecture arose from two movements. First was one of building highly cost effective microprocessor-based machines along the symmetric multiprocessing lines of the early multiprocessor vector machines. These new microprocessor-based machines were not positioned as supercomputers, since their CPUs were much slower than those of the vector machines, and the number of these processors available in a single machine (up to 36) was small compared to the number available in MPPs of that day (up to 512 or even 1024). However, these early machines were relatively low cost bus-based SMPs (symmetric multiprocessors), relatively easily programmed, and capable of supporting time-sharing (as opposed to MPP "space sharing") through the Unix operating system. These advantages translated into brisk sales, which placed these machines in the commercial mainstream and thus made them candidates for clustering.

As these microprocessor based SMPs appeared, many groups were building low cost computing systems out of clusters of single-processor workstations interconnected by Ethernet or FDDI networks. Eight or 16 machines in a cluster was a typical configuration. These low cost systems produced about the computing capability, for carefully structured application codes using message passing libraries, of a single vector processor of the day. None of these systems at that time could compete in capability with, say, a 16-processor Cray C-90 vector supercomputer. However, the same programming and network interconnection techniques could be used to build a cluster of SMPs. Members of our team worked with Silicon Graphics (SGI) in 1993 to do just that [1]. We built a cluster of 16 machines, each with 20 processors, interconnected in a 3-D toroidal topology by 20 FDDI rings. On our PPM code, this system delivered 4.9 Gflop/s sustained performance. To our knowledge, this was the first cluster system that delivered true supercomputer performance. Using a high-end, mainstream commercial system as the unit of replication in a cluster was the key to achieving supercomputer performance at relatively low cost, while still leveraging the software and inexpensive peripherals driven by the commercial market. Perhaps ironically, this idea has served the DoE's ASCI program well in procuring the most expensive and most capable systems now in use in the U.S. A simplified version of our PPM code, called sPPM [10], has been used as the performance benchmark for these ASCI program procurements, and as a result it has been run in 1998 at 1 Tflop/s sustained performance.

With the weight added in the U.S. by the ASCI program, it would seem that the SMP cluster architecture has become a clear winner. However, MPP fans still abound within the scientific computing community. We argue below that the true power of SMP cluster computing is the ease that it offers for dynamic load balancing in highly irregular computations. This potential still remains to be tapped in most of the classic areas of computational science. The exposition below will hopefully serve to inspire readers to go out and tap it for themselves, and through that effort to cement the gains that this new supercomputer architecture has by now achieved.

The SMP (or DSM) Cluster Programming Challenge:

Today's high-end computing platforms from U.S. vendors, DSM and SMP cluster systems, combine deep memory hierarchies in both latency and bandwidth with a need for many-hundred-fold to severalthousand-fold parallelism. Users of these systems have had to meet these challenges to efficient parallel program design armed only with minimal system software: Fortran, C, MPI, and support for POSIX threads on a network node. OpenMP is a promising new standard which can be used to generate portable code for a single DSM or SMP machine, but it does not address the cluster as a whole. Programming for dedicated cluster systems would be difficult enough, however DSM cluster systems are usually administered in order to optimize throughput, so that a mix of jobs of different sizes is set running at any given time. It is therefore very difficult to obtain for a single large job even an entire 128-processor machine, let alone a cluster of such machines. In order to run efficiently in this context, we restructured our PPM gas dynamics program so that it could dynamically adjust the number of processors it used on any single DSM machine of the cluster. This allowed the code to coexist with a time varying mix of other jobs. Our code's task manager continually entered requests into multiple job queues, adding any processors that were made available through this mechanism to the ongoing team. So far we have used this restructured code for problems with a regular structure, adapting to irregular processor loads from other users. However, it will be no additional difficulty to have this code dynamically adjust processor loads as a result of dynamically changing computational loads from regions of our own problem domain. The techniques that we have used to accomplish this code restructuring are outlined below. They are very general and should apply equally well to many other computational problems.

Hierarchical Shared Memory:

Our first versions of our PPM gas dynamics code, like the sPPM benchmark code that we wrote for the DoE's ASCI program, used thread-based shared memory multitasking within each DSM machine and MPI message passing over the DSM cluster. Not only was this hybrid coding technique clumsy, but it also made load balancing over the cluster network difficult. Following ideas presented in [7], our present approach extends to the entire cluster the shared memory multitasking approach that we use within each DSM machine. The key to this kind of parallel program is to decompose the work of the program into a sequence of tasks each of which requires only data from a restricted and compact data context and each of which can be executed from this data context without the need to communicate with any other task. This is a shared memory paradigm; tasks do not communicate directly with each other, but instead they read and write shared memory data structures. If we wish, we can think of the task's data context as including within it "messages" that have been written there by other tasks. This message passing through the intermediary of shared memory is, however, much simpler than message passing directly between ongoing processes. No message receiver ever need know the identity of the message sender, and vice versa. There is also no need for message buffering, since the data structures in shared memory are pre-allocated. It is still a good idea, of course, to write data needed by other tasks as soon as this data is generated and to read data supplied by other tasks at the latest possible moment. Synchronization of this form of message passing is therefore still required, although it is generally much simpler. In our code, we accomplish this synchronization by having each task set a semaphore variable in shared memory indicating when the task has been completed. (To avoid performance degradation due to false sharing, each semaphore must have its own cache line in memeory.) Tests on these task completion semaphores are performed before each task launch, so that a task once begun knows that it is safe to do all its work without further inquiry. The art of writing such programs is to order the list of tasks very carefully so that at any point in the program a very large number of tasks near that point in the sequence can be executing in parallel.

So far, the strategy of task parallelism outlined above should be familiar. This is the method for constructing parallel programs for SMPs. For DSM machines, there is one further detail. The non-uniform memory access of the DSM architecture forces the programmer to take the trouble of copying into the local memory of the executing processor elements of the task data context that will be over-written. Any intermediate data generated by the task that will not be written into shared memory for other tasks to read must also be placed in the local memory of the executing processor. In Fortran, this local memory placement is easily accomplished. The task is merely encapsulated in a subroutine (subroutine linkage is a negligible cost for any task with a hope of being efficient), and the data that must be local is dimensioned locally, so that it will be placed by the compiler on the subroutine stack (which is always in the best possible memory). This placement of the task work space in local memory eliminates the phenomenon of "false sharing" and greatly improves performance. Note that it is unnecessary to know which processor will execute the task. The subroutine stack will transparently be put in the right place.

Our program consists of a hierarchical set of task lists. The first set is a list of tasks intended for execution by entire machines of the cluster. These tasks, which we will call DSM tasks, are themselves composed of lists of smaller tasks, which we will call CPU tasks. A CPU task is encapsulated in a subroutine and is written for execution by a single CPU. DSM tasks are encapsulated in subroutines that are explicitly multithreaded for parallel execution. We make each task conform to a template consisting of three steps: (1) reading the task data context into the local, private task work space, (2) operating on the data context, and (3) writing data back into shared data structures. CPU tasks enjoy relatively large data bandwidth to the shared memory of their DSM machines, so they may perform these three task stages sequentially without significant loss of performance. (Machines that do not support this mode of task operation are difficult to sell and therefore are difficult to find.) However, DSM tasks do not enjoy high bandwidth access to the memory of other cluster members (or to shared disks). Therefore, steps 1 and 3 above must be overlapped with step 2. That is, the data for the next DSM task must be prefetched during the execution of the present DSM task. Also, the data produced by this DSM task must be written back to shared data structures during the execution of the next DSM task. This can be accomplished by encapsulating these data transfers in separate DSM tasks, with the obvious constraint that they must be executed by the same DSM that performs the real computational work of the DSM task to which they correspond. In our implementation, we have constructed memory server daemons that run on each DSM machine of the cluster and which asynchronously fulfill requests to "put" and "get" contiguous sections of arrays registered with them as globally accessible. We have coordinated the DSM tasks through a task manager process, which has only a single thread.

Hierarchical Shared Memory Parallel Implementation of PPM at NCSA:

We restructured the PPM gas dynamics code according to the hierarchical shared memory strategy outlined above. Memory server daemons were created to read and write a fast Fibre Channel network-attached disk system of our own design. Two 128-processor Silicon Graphics Origin-2000 machines at NCSA, interconnected by a single fast Ethernet, shared a common file system on 48 Seagate Fibre Channel disks supplied by LCSE industrial partner MTI. Each machine was connected to all 48 disks via 4 Fibre Channel loops. Each machine was connected to the disks through its own set of ports (the disks were dual ported). Read/write bandwidth from the PPM application from each machine was in excess of 270 MB/s, sustained, even when both machines accessed the disks simultaneously. Control information, such as DSM task completion semaphores, was passed via MPI over the fast Ethernet link.

This restructured PPM code was used to simulate Mach 2 homogeneous, compressible turbulence on a billion-cell (1024^3) uniform grid. A typical task for a single CPU was to update for a single 1-D pass a grid pencil of 4×4×256 cells. A typical task for a single 128-processor Origin-2000 machine was to update for six 1-D passes, or 2 time steps, a 256×256×512 brick of grid cells. The active data context for the job consisted of 32 old and 32 new grid brick records stored on the 864 GB shared Fibre Channel disk system. Each grid brick record of 954 MB consisted of 27 separate records: the brick interior (640 MB), 6 brick face records (27.5 MB each), 12 brick edge records (2.4 MB each), and 8 brick corner records (200 KB each). The active memory context in each participating DSM machine consisted of 5 grid brick records, or about 5 GB out of the 64 GB of DSM memory in each machine.

During each grid brick update, the Origin-2000 was asynchronously prefetching the next grid brick record and writing back the results of the previous grid brick update to 27 different grid brick records on disk. The grid brick record, after being read into DSM memory from disk (in 3.5 sec), was unpacked to form a single, augmented grid brick of 300×300×556 cells. This brick was then updated in six 1-D passes, with each consisting of 8192 single-CPU tasks (requiring 2.5 sec with 128 CPUs). In this demonstration run, there were barrier synchronization points at the ends of the 6 passes, but these can be eliminated at the cost of further code complexity. After the 6 passes, the new data was written into a new grid brick record in DSM memory, and this was transferred back to disk (in 3.5 sec).

Figure 1, below, documents about 4 days of continuous PPM computation at NCSA. Two 128-processor SGI Origin-2000 systems were used. Processors obtained by PPM on the first system are represented by the cream colored area in the figure. This system was not always available, due to scheduling of dedicated access for other jobs. Processors obtained by PPM on the second 128-processor system are represented by the blue area in the figure. PPM adjusted its number of processors on each machine at the beginning of each 1-D sweep for each grid brick. When 128 CPUs were in use, this

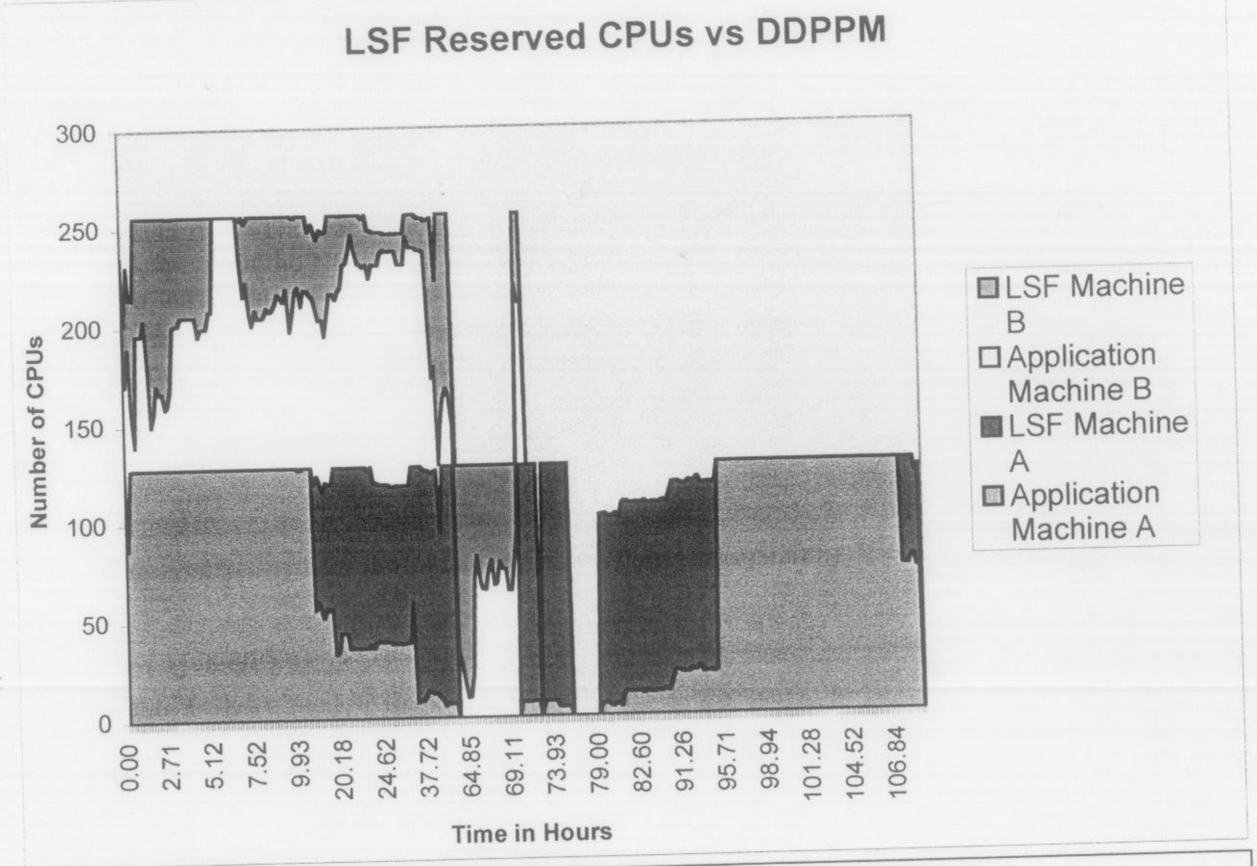

Figure 1. The usage of processors on two 128-processor Silicon Graphics Origin-2000 machines at NCSA is shown here over a 4-day period in the fall of 1998. The light blue and the tan areas of the diagram represent processors dynamically obtained by our PPM code for use on a single, billion-cell simulation of homogeneous, compressible turbulence. The other colors show processors devoted to other users of these systems. About half way through this period, one of the machines was given over to another user for a dedicated run. For a brief interval, both machines were unavailable to us. Brief periods in which processor utilization falls below 100% were caused by use of the machines by the system operators. The minimum response time of this PPM job to changes in resource availability was 2.5 seconds.

adjustment interval was about 2.5 seconds. Both machines were shared dynamically with other users, and PPM benefited by inserting requests for CPUs in several batch queues, grabbing the CPUs as they became available for this single, large computation. The small departures from full resource utilization that are shown reflect system functions performed by the operators, not any failure of PPM to exploit these opportunities.

Reevaluating this Demonstration Code for Parameters of the IBM SP System:

Our sPPM benchmark code indicates that, within a couple of per cent, the delivered performance of a PowerPC 604e microprocessor running at 333 MHz is equivalent to that of a MIPS R10K processor running at 195 MHz. Therefore, one SMP of 4 such PowerPC CPUs is about 32 times less powerful than a 128-CPU Origin-2000 for our PPM code. Hence we should reduce the computational labor involved in a single DSM task of the PPM code by about a factor of 32 in order to run well on this platform. We can do this by reducing the brick volume by factor of 8, resulting in a brick of $128\times128\times256$ cells, and by performing only a single 1-D sweep rather than 6 per task. This DSM task should now take $15\times32/48 = 10$ sec. Because we are performing only a single 1-D sweep, the data reuse in this task is 6 times less than on the Origin-2000. However, the ratio of "cluster" network bandwidth to DSM processing speed is now $4\times25/600 = 1/6$ Bytes/flop. This is 12 times greater than for the Origin-2000 at NCSA. As a result, the PPM code should run on this system even more efficiently. However, of course, in this model the entire problem data context must reside in the relatively expensive system memory rather than on relatively inexpensive Fibre Channel disks. A form of overhead for the parallel code is redundant computation performed in "ghost" cells surrounding each grid brick. The fraction of the computation time devoted to this redundant work would have remained the same as in the NCSA run if we had performed three rather than just one sweep per DSM task. We have thus reduced the redundant computation overhead by a factor of $(278^2\times534 - 256^2\times512)/(8\times(130^2\times263 - 128^2\times256)) = 3.9$ and the efficiency of the job should therefore be even greater. We note that these projections are merely educated guesses, and they do not incorporate any consideration of problem I/O. They nevertheless suggest that our parallel programming model might be portable to systems with fairly dramatically different parameters than the Silicon Graphics Origin cluster at NCSA.

Reevaluating this Demonstration Code for Parameters of the HP System:

We presently await results of tests that are underway with the cooperation of HP personnel and of the computing center at Caltech.

Assumptions that Permit Efficient DSM Cluster Programs:

It is tempting to speculate that perhaps all codes aimed at the simulation of physical systems on grids could be implemented in the above fashion. As a result, we have attempted to abstract those characteristics of our PPM code that we feel are essential in enabling this restructuring. We state them here in the form of assumptions that we believe are necessary for such hierarchical shared memory parallel program execution. First, we assume that a job can be decomposed into a set of tasks that can be executed independently, so long as certain previous tasks are completed at task launch. We further assume that each task can be made to conform to a model, or template, in which:

1) possibly remote data is copied into local memory,
2) this data is operated upon mightily,
3) a few results are written back to possibly remote storage.

We assume that the tasks can be constructed so that, in general, the larger the data context for the task, the larger the amount of potential data reuse. This assumption is necessary to accommodate low cluster bandwidths. To accommodate large cluster latencies we must also assume that the task data contexts can be constructed so that they may be read or written back in only a small number of sequential data transfers. Once in fast local memory, these data contexts can be efficiently reorganized if necessary.

We assume that global barriers (Amdahl's Law) can be avoided by providing greater system resources and/or by minor modifications of the numerical algorithm. Examples of this principle abound. For example, a program may require all processing to stop so that an image of the problem can be written to a restart file on disk. However, if additional system memory is provided, this restart dump can be written asynchronously without impeding the program flow. Another program might require that a global reduction operation be performed after a time step is completed in order to determine the value of the next time step. However, if enough memory is provided to store the previous problem state, we may guess the time step value (the minimum of the previous 25 time steps might be a good guess) and proceed speculatively. In the rare event that we guess badly, the saved system state will permit us to recover. As a final example, we may be performing an implicit calculation that appears to require global information to be assembled in order to update the value of a variable at a single spatial location. By revising the numerical algorithm slightly, we could require up-to-date information only for the local region and use information from the previous time step or iteration for the more distant data. Once again, this would require a commitment of additional system memory to the job.

Feasibility Requirements:

There are bandwidth requirements for the cluster network, but essentially no meaningful latency requirements. The bandwidth requirements are determined by the demand that any computing resource should be able to execute any task, regardless of the location of its data context. Data prefetching and asynchronous write back are absolutely essential. The task manager should have limited intelligence to avoid stupid data movement. It should dynamically reorder the task list, permuting elements that are equally or nearly equally qualified candidates for the next task to be launched, taking data location over the network into account. Finally, local memory for various computing resources must be sufficient to accommodate data contexts offering sufficient data reuse, but this is not a new requirement.

Relation to Other Work:

Many investigators have been concerned in recent years with enabling shared memory programs to execute on cluster systems. Some of the early work in this area led directly to the development of distributed shared memory (DSM) machines, particularly the work of the Stanford team led by John Hennessy (cf. Kuskin et al. 1994). More recent contributions (see reference list) have focused on software systems that create from a cluster of machines the practical effect, rather than simply a research prototype, of a DSM machine. Much of this work, although not all, has focused on clusters of single-processor machines and therefore features only a single level of the two-level shared memory discussed in the present work. Also, much of the work has involved very fine granularity of software shared memory access, often based on machine memory pages. In this respect this work stands in contrast to the approach advocated here. Some of the recent work on out-of-core computation algorithms, particularly that of Salmon and Warren (1997) and Nieplocha and Foster (1996) involves concepts relevant to the work presented here. A feature that this out-of-core work, particularly that of Salmon and Warren, shares with our own is the emphasis on restructuring the numerical algorithm to function well in the new mode. The work presented here is based on ideas set out in Woodward (1996). Continually updated lists of references to work on SMP cluster computing can be found on the "clumps home page," at http://now.cs.berkeley.edu/clumps, and through links maintained on that page.

Scientific Results:

For over a decade, we have been pursuing every available opportunity to use powerful new computing hardware in order to enhance the faithfulness to nature's complexity of our simulations of turbulent fluid flow. We report below some of the new 3-D simulations of compressible turbulent fluid flow that we have performed with our PPM gas dynamics code on SMP and DSM cluster platforms. These include simulations of convection in the envelopes of pulsating and rapidly rotating model stars, of chunks of homogeneous, compressible turbulence, and of the unstable acceleration of fluid interfaces by shocks.

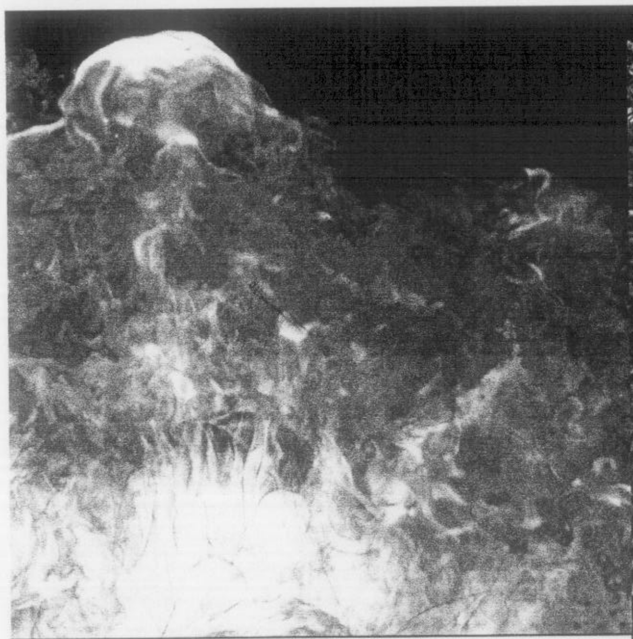

Figure 2A. A perspective volume rendering of the distribution of entropy near a small section of the unstable interface near the end of the sPPM simulation of the Richtmyer-Meshkov instability. The entropy of the shocked denser gas is shown as white, while that of the shocked, more diffuse gas is transparent. The region of turbulent mixing is in the green region of this "forest of broccoli."

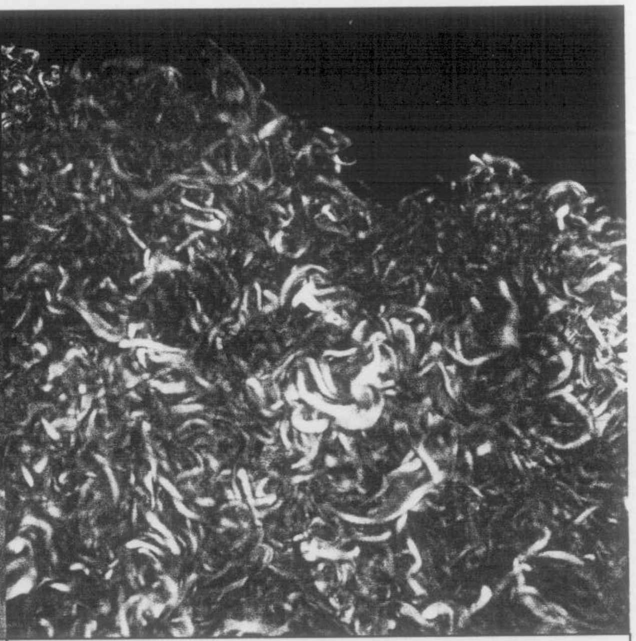

Figure 2B. A perspective volume rendering of the same region near the unstable interface as is shown in the figure at the left. Here the magnitude of the vorticity is visualized. This twisted collection of vortex tubes is characteristic of fully developed turbulence, a conclusion that is supported by the velocity power spectra shown in Figure 3. Note the absence of strong vorticity along the top of the "mushroom cap."

The power of SMP clusters has enabled us during the last two years finally to resolve a reasonable range of turbulent scales within the context of an interesting larger flow. The largest such calculation that achieves this goal is a simulation of the Richtmyer-Meshkov instability of a shock-accelerated fluid interface. This simulation was performed at Livermore with the simplified version, sPPM, of our PPM code running on a grid of 8 billion uniform cells. Because this sPPM code version was used as the ASCI Blue platform procurement's official SMP cluster benchmark, we had the expert assistance of Steve White and his colleagues at IBM. They handled the details of communication between the two (of three) 488-node (1952-processor) "boxes" in Livermore's IBM SP system for this special code implementation. Steve White also collaborated with IBM's compiler group to produce outstanding performance for this code on the entire 5856-processor system. By artfully counting the floating point operations that can be associated (legally, in the sense of the ASCI procurement) with the reciprocal and square root functions, the sustained aggregate performance was revealed to be in excess of 1 Tflop/s. Regardless of how one counts, this code's performance on the 5856-processor IBM system is exceptional.

The Richtmyer-Meshkov instability results when a shock accelerates an interface separating two

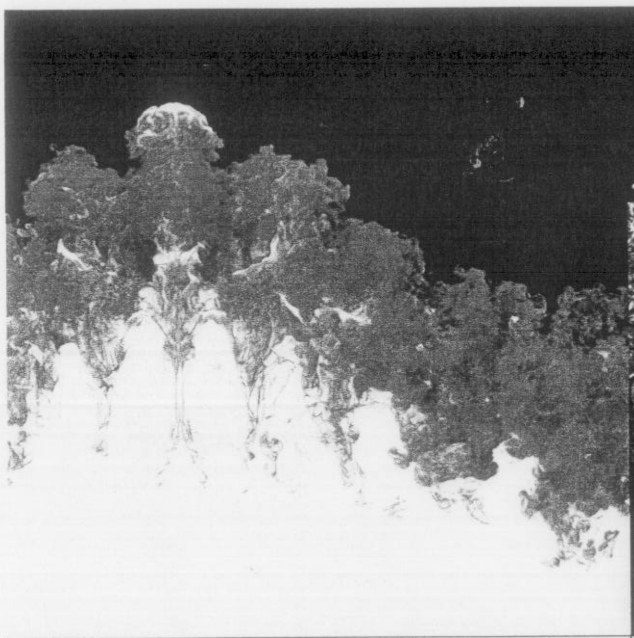

Figure 2C. A perspective volume rendering of the distribution of entropy near a thin slice of a small section of the unstable interface near the end of the sPPM simulation of the Richtmyer-Meshkov instability. The entropy of the shocked denser gas is shown as white, while that of the shocked, more diffuse gas is transparent. The region of turbulent mixing is in the green region of this "forest of broccoli."

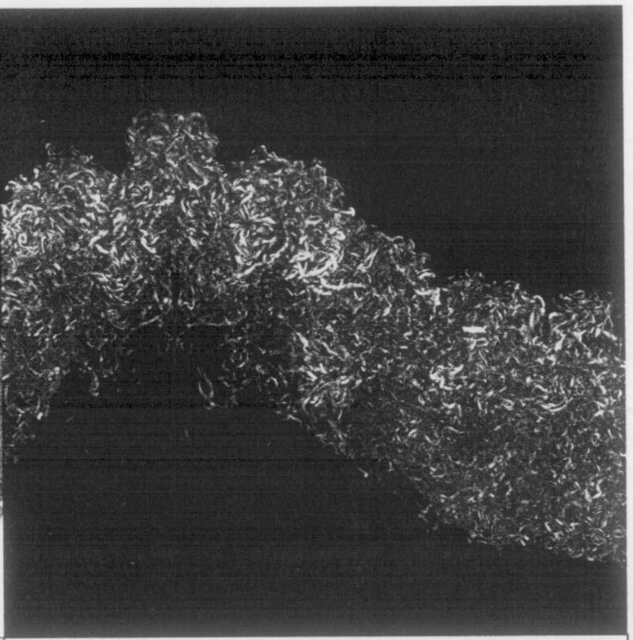

Figure 2D. A perspective volume rendering of the same region near the unstable interface as is shown in the figure at the left. Here the magnitude of the vorticity is visualized. This twisted collection of vortex tubes is characteristic of fully developed turbulence, a conclusion that is supported by the velocity power spectra shown in Figure 3. Note the absence of strong vorticity along the top of the "mushroom cap."

fluids of different densities. Because the shock travels more slowly in the denser medium, perturbations in the interface location cause sections of the shock to lag slightly behind the shock's average position. The oblique sections of the shock which therefore develop set up transverse motions in the compressed gas behind it which are focused on the regions behind the sections of the shock which lag. These focused motions increase the pressure in these regions, and hence the shock strength and propagation speed there. As a result, the shock straightens out, but it leaves behind transverse motions which are converted to longitudinal ones due to the jump in the sound speed at the fluid interface, where the denser material squirts out into the less dense one. These longitudinal motions are not further accelerated, but they are not damped either. Shear at the interface subsequently leads to the development of turbulence and (in the absence of surface tension) to a mixing of the two fluids on small scales.

We initialized our simulation of this instability with a planar Mach 1.5 shock approaching through the less dense gas ($\gamma = 1.3$) a perturbed interface with a gas 4.88 times denser (also with $\gamma = 1.3$). The perturbation of the interface was $0.01\ [-\cos(2\pi x)\cos(2\pi y) + |\sin(10\pi x)\sin(10\pi y)|]$. We initialized the fluid interface as a smooth transition spread over a width of 5 grid cells. This initialization greatly reduced the amplitude of the high frequency signals that are unavoidable in any grid-based method. The sPPM method of capturing and advecting fluid interfaces forces smearing of these transitions over about 2 grid cells and resists, through its inherent numerical diffusion, development of very short wavelength perturbations. By setting up the initial interface so smoothly, we assured that after its shock compression it would contain only short wavelength perturbations that the sPPM scheme was designed to handle, and that none of these perturbations resulting only from the discretization of the problem on a grid would be mistaken by the scheme for real signals. Nevertheless, the problem is physically unstable, so one cannot be too careful in interpreting the results. We are guided in our interpretation by a series of coarser grid

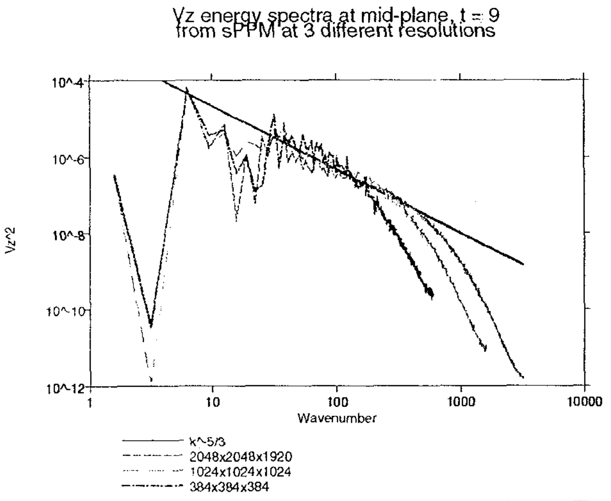

Figure 3. Longitudinal velocity power spectra.

simulations of the same problem and by the data shown below. The main point we would like to stress here is that on this grid of 8 billion cells, we are finally at a point where we are able to resolve in this single computation both the primary, long wave-length behavior of the Richtmyer-Meshkov instability and also the secondary, short wavelength behavior of the turbulence that grows out of the shear which this instability produces.

This sPPM simulation was carried out on a brand new computing system, which therefore did not possess all its planned support systems of disks, archival storage, and visualization hardware. As a result, we were able to archive only 10 full-information snap shot files, each compressed by a factor of 2 to represent each number in only 16 bits. Each of these files was 84 GB in size, so this is still nearly a TB of information. However, based upon preliminary runs of this same problem at lower grid resolution, we determined a single variable, related to the entropy of the gas, and a scaling of this variable to 255 intensity levels, that we wished to save in more complete form. Each snap shot for this single entropy variable was only 8 GB, which made it possible to archive

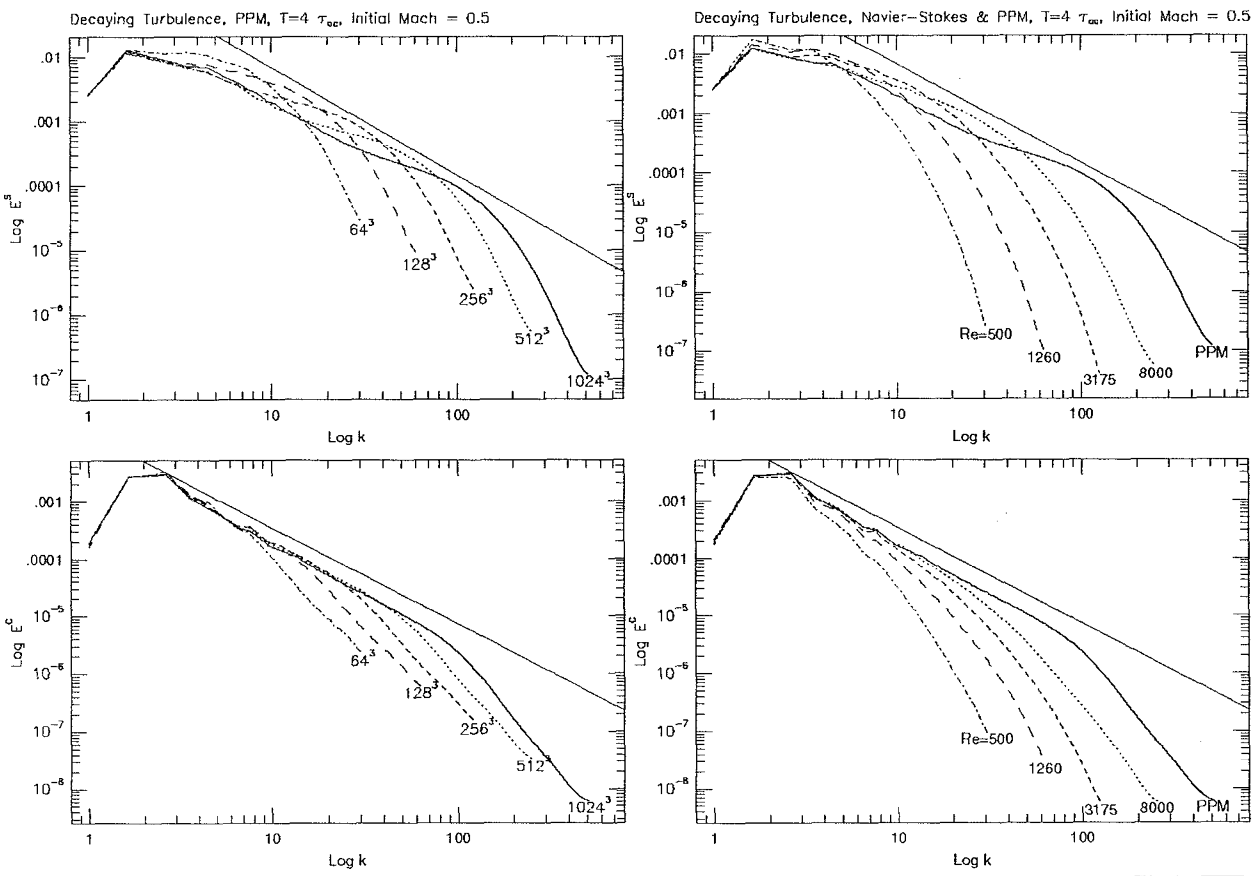

Figure 4A,B. Solenoidal (top) and compressional (bottom) velocity power spectra of decaying turbulence.

274 such snap shots during the course of the simulation. We may thus ask any question about the dynamics of the gas entropy in this run and have a good hope of an answer, but questions about other fluid dynamic state variables must be restricted to their behavior as shown only in the 10 larger snap shots which we were able to preserve. Such constraints are typical of computations carried out on first-of-a-kind computing systems like this one.

In Figures 2A&C and 2B&D above, the entropy and the magnitude of vorticity are visualized in the same small region near the center of the unstable interface near the end of the simulation. From these figures we can see that this simulation has indeed captured both the macroscopic and the microscopic scales of the Richtmyer-Meshkov instability and its secondary Kelvin-Helmholtz instabilities. We can make this statement quantitative by looking at the velocity power spectrum within a plane slicing through the unstable layer. Such a power spectrum is shown in Figure 3. Actually, in this figure three such spectra are compared. The three spectra come from simulations on progressively finer grids of 384^3, 1024^3, and $2048^2 \times 1920$ cells. The low frequency parts of these spectra reflect the initial perturbations and the harmonic modes that their nonlinear interactions have produced. To the right of the higher frequency initial perturbation, a short section of a turbulent inertial range, with a Kolmogorov power-law slope of

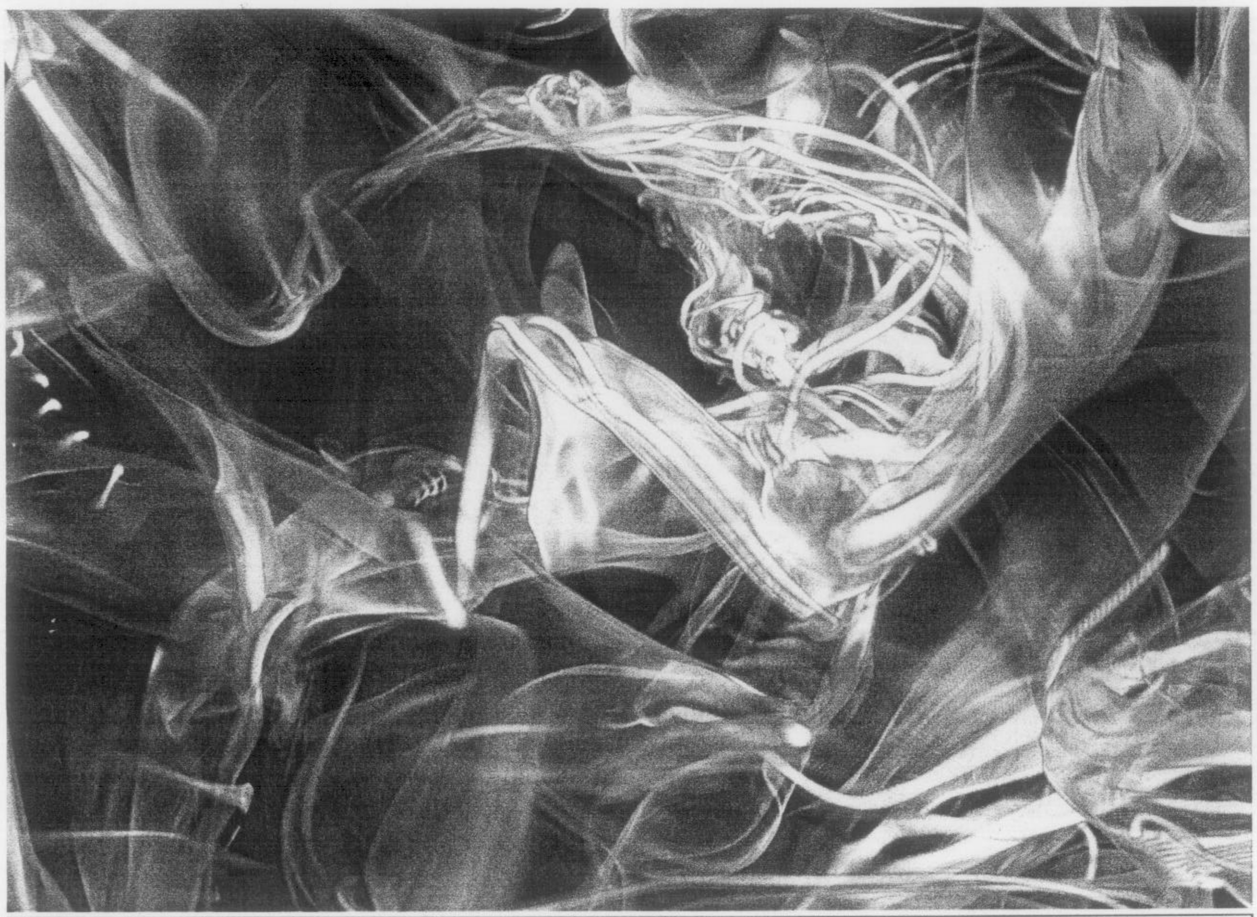

Figure 5. The distribution of the magnitude of vorticity is visualized in this volume rendering of a region of the billion-cell simulation of homogeneous, Mach ½ turbulence. Near the center of the figure, a group of thin vortex tubes are spiraling around each other as a whole section of a vortex sheet rolls up. At the left, a series of roughly parallel vortex tubes is intensifying out of another vortex sheet. This visualization captures the developing turbulent flow in the midst of its transition from a state dominated by vortex sheets to one in which such sheets can no longer be identified within the dense tangle of spaghetti-like vortex tubes.

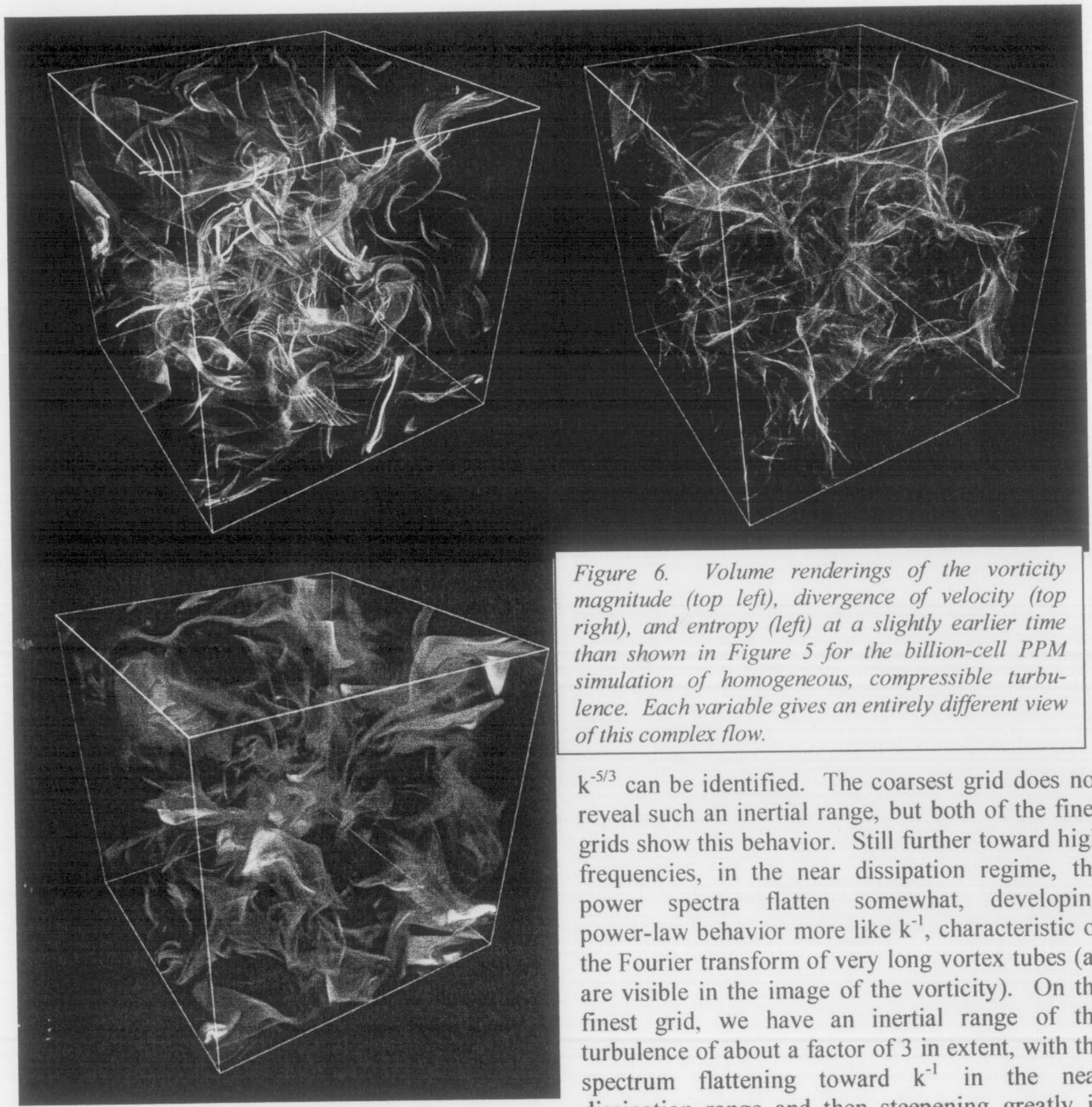

Figure 6. Volume renderings of the vorticity magnitude (top left), divergence of velocity (top right), and entropy (left) at a slightly earlier time than shown in Figure 5 for the billion-cell PPM simulation of homogeneous, compressible turbulence. Each variable gives an entirely different view of this complex flow.

$k^{-5/3}$ can be identified. The coarsest grid does not reveal such an inertial range, but both of the finer grids show this behavior. Still further toward high frequencies, in the near dissipation regime, the power spectra flatten somewhat, developing power-law behavior more like k^{-1}, characteristic of the Fourier transform of very long vortex tubes (as are visible in the image of the vorticity). On the finest grid, we have an inertial range of the turbulence of about a factor of 3 in extent, with the spectrum flattening toward k^{-1} in the near dissipation range and then steepening greatly in the dissipation range (for wavelengths of about 8 grid cells or less). Our experience with simulations of homogeneous turbulence (see below) indicates that the indirect effects of the dissipation of the turbulent motions by the numerical viscosity of the sPPM scheme extend no further toward long wavelengths than the short section of k^{-1} slope. Therefore, presumably, this simulation properly incorporates the physical effects of the interactions of the original perturbation scales and their first few harmonics with the smaller scales of the turbulence that the instability has generated.

Concentrating all the DSM Cluster Computational Power on a Single Chunk of Turbulence:

Using the Silicon Graphics DSM cluster at Los Alamos in the spring and summer of 1997, we were able to simulate a small chunk of homogeneous, compressible turbulence. With the PPM gas dynamics code running with periodic boundary conditions on a uniform grid of 1024^3 cells, we achieved a fully developed turbulent flow containing a Kolmogorov inertial range of about a factor of 8 in wavenumber.

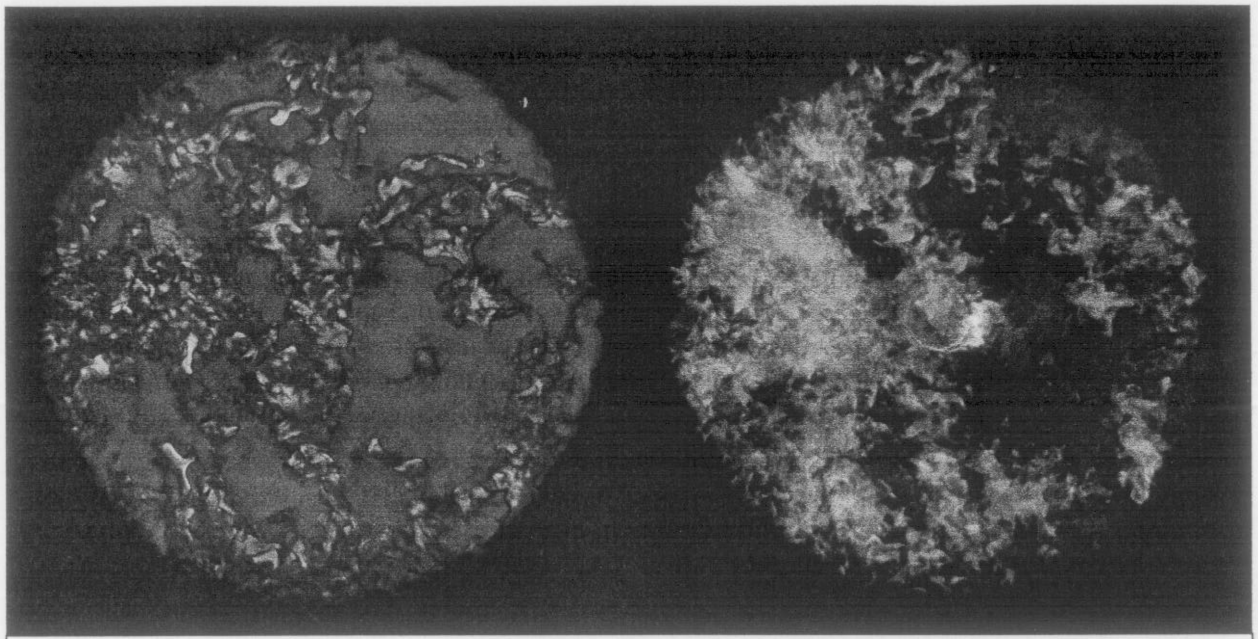

Figure 7A,B. Two perspective volume renderings of a PPM simulation of the convective envelope of a model giant star with 3 times the mass and 4500 times the luminosity of the sun. 80% of the mass of this giant star is in the extended convective envelope shown here, and 20% is in a dense, hot stellar core which is stable to convection. For such a star, the envelope radius is comparable to the orbit of the earth, while the core radius is comparable to the radius of the earth. Here the core is artificially enlarged, while still very small, so that its radius of a tenth that of the envelope can be resolved on the computational grid of 512^3 uniform cubical cells. Temperature fluctuations relative to average values on isopressure surfaces are shown, with red and yellow depicting warm and hot temperatures and blue and aqua representing cool and cold temperatures. At the left, the envelope is made relatively opaque, so that the surface pattern of convection cells is visualized. At the right, the gas has been made relatively transparent, so that the interior strong dipolar flow pattern is revealed. Clearly, the left half of this stellar envelope is cooler than the right half. The heated gas which has flown over and around the core is evident as the yellow stream at the right.

This flow was initiated with a statistical sample of long wavelength velocity disturbances centered fairly narrowly on a wavelength of half the periodic scale. The density and pressure were constant in the initial state, and the amplitude of the initial velocity disturbances gave an rms velocity perturbation of half the sound speed. The velocity power spectrum is shown in figure 4A at a point during the decay of this flow when the turbulence has become fully established. This spectrum is compared in the figure with those of identical runs with the PPM code on grids of progressively coarser resolution: 512^3, 256^3, 128^3, and 64^3. The velocity power spectra are shown in this figure for both the incompressible (solenoidal) and compressible components of the velocity field (upper and lower panels, respectively).

The convergence of the power spectra is clear. Each successively finer grid leaves the long wavelength behavior invariant while adding another section to the spectrum at high wavenumbers. At the right-hand end of each power spectrum is of course a short region of strong damping due to the numerical dissipation of the PPM Euler scheme. For the solenoidal power spectra, this picture is complicated by the presence in the near dissipation range of a section of flatter slope, about k^{-1}, extending over roughly a factor of 4 in wavenumber. This behavior was noted first in our numerical simulations several years ago, and it has been confirmed in other simulations as well as in experiments since then. This behavior can also bee seen in the spectra in Figure 3. Interestingly, it can also be seen in the power spectra in Figure 4B, which come from identical PPM runs in which the Navier-Stokes dissipation terms have been included [11]. A Prandtl number of unity was chosen, and the Navier-Stokes viscosity coefficient was chosen by the demand that when these coefficients remain fixed and the computational grid is refined by a factor of 2 in each dimension, the power spectra remain unchanged. This demand of convergence of the power spectra can of course not be applied in the Euler case, since a grid refinement reduces the effective viscosity of the flow by a factor of 8 (for PPM). The Navier-Stokes runs were carried only to a grid of 512^3. at which resolution there is still no Kolmogorov inertial range at all for this flow. Nevertheless, the trend of these curves — for grids of 64^3, 128^3, 256^3, and 512^3 — indicates that with a grid of about 4096^3 we would see such a region of the spectrum with about the same extent in wavenumber as the Euler approach gives us with only 1024^3 cells.

The billion-cell Mach ½ turbulence calculation gives an enchanting, detailed view of the development of a turbulent flow. The instability of originally smooth vortex sheets to form systems of vortex tubes and the subsequent braiding of these tubes about each other is clearly seen. Figure 5, above, gives a glimpse of this fascinating process when it is about half-way along, and Figure 6 shows volume renderings of three variables, the vorticity magnitude, the divergence of the velocity, and the entropy, a bit earlier in the simulation.

Simulating Turbulence Driven by Convection, which in turn Interacts with Stellar Pulsation:

In a number of PPM simulations using NCSA's Silicon Graphics DSM cluster, we have explored the interaction of turbulence with thermal convection, and of this turbulent convection in turn with either rotation or pulsation. This work began with our participation in an NSF-funded Grand Challenge team project, together with the team of Juri Toomre at the University of Colorado, Boulder. The computational power of the DSM cluster made it possible to extend the grand challenge work on small 3-D blocks representative of the solar convection zone near its surface to include the entire model star. Our first simulations, begun in the fall of 1996, used a model star which was chosen to explore the interaction of convection with rotation. This calculation incorporated a treatment of the free surface of the star which used a multifluid algorithm based on the SLIC algorithm of Noh and Woodward [12], but with gravity serving to make the interface between the stellar gas and the second "fluid" (vacuum) stable. The escape of heat through this surface was treated rather crudely, however, with the time-averaged surface heat flux forced to match the constant rate at which heat was introduced into the stable central region of the model star.

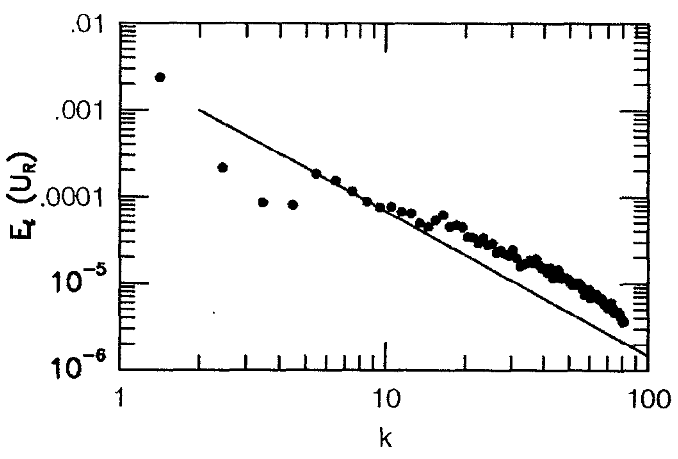

Figure 8. The radial velocity power spectrum from the middle of the convective envelope of our model giant star is here analyzed in terms of spherical harmonic modes. Note the dominance of the dipole mode and the Kolmogorov power-law behavior at higher wavenumbers (indicated by the straight line). The slight flattening of the spectrum toward the highest wavenumbers, just before the dissipation range, is a feature we have seen in every compressible, turbulent flow we have simulated.

In the fall of 1997, we used the same multifluid, 3-D Cartesian PPM code at NCSA to simulate pulsating stars. In our most recent simulation, we chose a luminous model star to drive very rapid, vigorous convection. In this case, we set up the unperturbed stellar model without rotation but with a very deep convection zone as is found in giant stars. In order to resolve it on the computational mesh, we enlarged the convectively stable, hot stellar core to about 10% of the stellar radius. In a typical giant star, the hot, stable core would have a radius comparable to that of the earth while the convective envelope around it would have a radius comparable to that of the orbit of the earth about the sun. We artificially reduced this tremendous dynamic range in our simulation, keeping the stellar core small but still resolvable. In Figure 7, above, we see two volume renderings of our simulated giant star convective envelope. At the left, we have made the envelope relatively opaque, so that we see the surface features, while on the right we have made the envelope sufficiently transparent that we may see right through it to the hot stellar core within. In both renderings, we show the temperature fluctuations relative to average temperatures on each isopressure surface. Red and yellow represent warm and hot temperatures, while blue and aqua represent cool and cold ones.

Aside from the pulsation of this model giant star envelope over a range of about $\pm 10\%$ in radius, the biggest surprise of this calculation, as with our first such simulation in 1997, was the prominence of global modes of convection. Treatments of convection for evolution calculations for stars of this type assume that the convection can be characterized by mixing length models which involve only local parameters. However, this simulation reveals a propensity for global modes of convection. In fact, animated sequences of images like these reveal a general dipolar flow pattern, with relatively cool gas descending toward the core from the left in Figure 7B, becoming heated while passing over the core at about a fifth of the local sound speed, and rising as relatively warm gas to the right in the figure.

The deep convection in the model giant star envelope that we have been discussing has provided us with a fully developed thermal convection flow with 5 pressure scale heights over which the special conditions of either the upper or the lower boundaries exert no major effects. In this flow the periodic boundary conditions in the "horizontal" directions are not at all artificial, but instead simply express the overall spherical geometry of the problem. Here we have, as in the Richtmyer-Meshkov problem, turbulence driven by a real physical process which is itself simulated in complete detail within the calculation. In contrast to the Richtmyer-Meshkov example, however, this thermal convection flow is statistically steady, except of course for the periodic radial pulsation of the envelope as a whole. This PPM simulation on a 512^3 grid was carried out over about 20 pulsation periods, so that the flow has had a good deal of time to relax. Velocity power spectra of the flow near the middle of the envelope (middle values of the radius) are shown in Figure 8. Once again, the longest wavelength modes are characteristic of the driving physical process and there is a turbulent Kolmogorov inertial range, in this case rather short. Again there is a flattening of the power spectra in the near dissipation range followed by a dramatic steepening as the turbulence is dissipated at the highest wavenumbers. In this simulation, not only is the nonlinear interaction of the small scale turbulence with the convection flow treated in detail, but the nonlinear interaction of the dipolar convection flow with the global radial pulsation is accounted for as well. We are beginning a new series of such giant star simulations incorporating more realistic models of the gas equation of state, including gas ionization effects, and improving our treatment of the escape of heat from the stellar surface.

Conclusions:

The new generation of powerful DSM and SMP cluster computers enables simulations of fluid dynamics at sufficient resolution to compute the complex nonlinear interactions of small-scale turbulent motions within a large-scale driving flow. With a new programming model of hierarchical shared memory multi-tasking, it is possible to exploit these new systems without disrupting the flow of small and medium-sized jobs that makes their existence possible.

Acknowledgements:

We would like to acknowledge generous support for this work from the Department of Energy, the National Science Foundation, and NASA. Our work on implementing codes efficiently on DSM clusters began in 1993 with support from Silicon Graphics and the Army Research Laboratory and has continued with support from the DoE's ASCI program through a Level-2 project with Los Alamos, subcontract B33700016-3Y, completed in 1998, and with support from NSF through a MetaCenter Regional Alliance grant, ASC-9523480, and support from the PACI program through subcontracts from NCSA. Support for our continued numerical algorithm development and the packaging of our algorithms in library form, PPMLIB, has come from the DoE's Office of Energy Research through grants DE-FG02-87ER25035 and DE-FG02-94ER25207, respectively. Support for our investigations of turbulent compressible convection has come from an NSF Grand Challenge Application Group award, ASC-9217394, through a subcontract from the University of Colorado, and more recently from a NASA Grand Challenge team award, NASA Cooperative Agreement Number (CAN) NCCS-5-151, through a subcontract from the University of Chicago. We would also like to acknowledge support for computer time from the ASCI program at Livermore and Los Alamos, from the NSF PACI program through NCSA, and also local support from the University of Minnesota's Minnesota Supercomputing Institute. We would also like to acknowledge support to this project for scientific visualization and handling of the very large data sets involved. This support has come from an NSF CISE Research Infrastructure grant, CDA-950297, from the NSF's PACI program through NCSA, from the DoE ASCI program at Livermore (through NCSA), and from the LCSE industrial partners Silicon Graphics, MTI, Ciprico, Seagate Technology, Ancor, and Brocade Communications. We made use of powerful visualization machines at Livermore and at the ACL in Los Alamos together with the LCSE visualization systems to process the data from the large runs performed at these labs. This work was performed in part under the auspices of the U.S. DoE by the Lawrence Livermore National Laboratory under contract NO. W-7405-ENG-48.

References:

1. Silicon Graphics and the University of Minnesota, "Grand Challenge Computing, A No-Frills Technical Documentary," a video documentary produced by Silicon Graphics in October, 1993, describing our group's collaboration with them in Sept., 1993, to build and demonstrate the first SGI Challenge Array prototype.
2. Woodward, P. R., and Colella, P., 1984. "The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks," *J. Comput. Phys.* **54**, 115-173.
3. Colella, P., and Woodward, P. R., 1984. "The Piecewise-Parabolic Method (PPM) for Gas Dynamical Simulations," *J. Comput. Phys.* **54**, 174-201.
4. 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.
5. 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.*
6. Woodward, P. R., 1994. "Superfine Grids for Turbulent Flows," *IEEE Computational Science & Engineering*, Vol. **1**, No. 4, pp. 4-5+cover illustration, (December, 1994).
7. Porter, D. H., Woodward, P. R., and Pouquet, A., 1998. "Inertial Range Structures in Decaying Turbulent Flows," *Physics of Fluids* **10**, 237-245.
8. 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.*
9. Woodward, P. R., "Trade-offs in Designing Explicit Hydrodynamic Schemes for Vector Computers," in *Parallel Computations*, Academic Press, 1981.
10. Woodward, P. R., and Anderson, S. E., 1995. SPPM benchmark code, LCSE version. Available at *http://www.lcse.umn.edu.*
11. Sytine, I. V., Porter, D. H., Woodward, P. R., Hodson, S. W., and Winkler, K.-H., 1998. "Convergence Tests for Piecewise Parabolic Method and Navier-Stokes Solutions for Homogeneous Compressible Turbulence," submitted to *J. Comput. Phys.*, also available as Minnesota Supercomputing Institute Technical Report UMSI 98/169, Oct., 1998.
12. 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)

13. W. F. Noh and P. R. Woodward, "SLIC, Simple Line Interface Calculation," *Lecture Notes in Phys.* **59**, 330 (1976).
14. Baden, S. B., and Fink, S. J., 1999. "A Programming Methodology for Dual-Tier Multicomputers," to appear in *IEEE Transactions on Software Engineering.* (available at http://www-cse.ucsd.edu/groups/hpcl/scg/kelp.html)
15. 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.
16. Cox, A. L., Hu, Y. C., Lu, H., and Zwaenepoel, W., 1999. "OpenMP on Networks of SMPs," to appear in Proc. 13th International Parallel Processing Symposium, April, 1999. (available at http://www.cs.rice.edu/~willy/TreadMarks/papers.html)
17. Culler, D. E.. Arpaci-Dusseau, A., Arpaci-Dusseau, R., Chun, B., Lumetta, S., Mainwaring, A., Martin, R., Yoshikawa, C., Wong, F., 1997. "Parallel Computing on the Berkeley NOW." JSPP'97 (9th Joint Symposium on Parallel Processing), Kobe, Japan. (http://now.cs.berkeley.edu/Papers2/Postscript/jpps.ps)
18. 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://www-flash.stanford.edu/architecture/papers/paperlinks.html)
19. 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)
20. Gropp, W. W., Lusk, E. L., 1995. "A Taxonomy of Programming Models for Symmetric Multi-processors and SMP clusters," in Proceedings of Programming Models for Massively Parallel Computers, October 1995, pp. 2-7. (HTML Abstract: http://now.CS.Berkeley.EDU/clumps/gropp.html)
21. Kuskin, J., Ofelt, D., Heinrich, M., Heinlein, J., Simoni, R., Gharachorloo, K., Chapin, J., Nakahira, D., Baxter, J., Horowitz, M., Gupta, A., Rosenblum, M., and Hennessy, J., 1994. "The Stanford Flash Multiprocessor," Proc. 21st International Symp. On Computer Architecture, pp 302-313. (available at http://www-flash.stanford.edu/architecture/papers)
22. Lumetta, S. S., Mainwaring, A. M., Culler , D. E., 1997. "Multi-Protocol Active Messages on a Cluster of SMPs," Proc. Supercomputing '97. (available at http://now.cs.berkeley.edu/clumps/sc97)
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.