A PERSPECTIVE ON SUPERCOMPUTING: THREE DECADES OF CHANGE

Paul R. Woodward

University of Minnesota

Laboratory for Computational Science & Engineering

August 31, 1996

I. INTRODUCTION

It has been my great good fortune to have done my work in computational science on supercomputers from the time I was starting out in graduate school 28 years ago to the present. This article gives a perspective on supercomputing based upon these 28 years in the field, but of course supercomputing is a broad subject, and mine will be only one account of it. I should therefore begin by revealing my biases. These come mainly from a concentration upon computational fluid dynamics, and, more specifically, in the compressible flow regime where time dependent problems play a major role. The combination of time dependence and the requirement of three dimensionality to properly represent the important phenomenon of fluid turbulence have been sufficient to stress the world's most powerful computing systems. These problems have made equal demands upon the disk and tape storage systems, to save the data generated, and upon scientific visualization systems, in order to translate that data into scientific understanding.

Much of my effort in this field has been devoted to the design and implementation of numerical algorithms to carry out the supercomputer computations. As a result, much of my direct experience of the various supercomputers I have used consists of the programming efforts required to make them perform. This account therefore focuses upon the changes over the years in the design of these machines and the corresponding changes demanded of the programs which ran on them. These issues of supercomputer design and its interaction with numerical algorithm design and implementation are especially important today, when several different approaches and programming models are in use with none, as yet, occupying a dominant position.

This article will not attempt to include a comprehensive account of all the principal supercomputers and supercomputer designs of the last 3 decades. Nevertheless, I should point out that I have had the pleasure and the privilege to compute on the CDC 6600, the CDC 7600, the CDC Star-100, the Cray-1, the Cray-XMP, the Cray-2, the Connection Machines CM-2 and CM-5, the Cray T3D, and the Silicon Graphics Challenge Array and Power Challenge Array. All these machines, except the Cray-1 and the CM-2, were serial #1. This series of first-of-their-kind supercomputers covers 3 decades and 3 great revolutions in supercomputer design — the introduction of vector computing, the introduction of parallel processing on multiple CPUs, and the introduction of supercomputing on microprocessors with cache memories and arranged in a hierarchy of clusters. The supercomputing community is in the midst of the third revolution at the moment, and therefore some in the community may not agree that it has in fact begun.

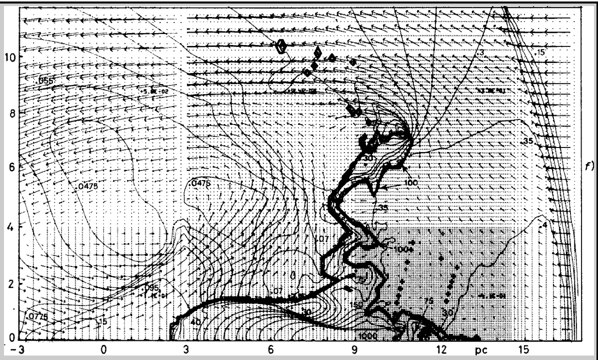
In this article I will give an account of the 3 great changes in supercomputer design just mentioned and the impact which they have had on the way programs are written and on the choice of numerical algorithms for those programs. Using the fluid dynamics familiar to me, I will illustrate what kinds of computations these supercomputers could and can do. Finally, I will offer my view of where this all is leading, with the understanding that the present pace of change makes such predictions by anyone fundamentally unreliable.

II. VECTOR SUPERCOMPUTING

Vector supercomputing began on the CDC 7600 at Livermore. This may come as a surprise to readers old enough to remember that the 7600 was not a vector machine. However, the first vector machine, the CDC Star which the Livermore lab had ordered in 1967, was delivered 7 years late. Somewhere around the middle of that interval Frank McMahon at Livermore realized that the 7600 could emulate a vector supercomputer in software, and in the process run vector codes around 4 times faster than they would run otherwise. The key to this realization was that the large core memory of the 7600, half a million 60-bit words, or roughly 4 MB, fetched data in units of 8 words, whether the program used all 8 words or not. If a tight loop in the instruction stack could be written which would use all this data as it arrived, the latency of the memory system could be hidden. This is the fundamental idea of vector computation. Pipelined functional units are kept busy by bringing from memory long sequences of data values each of which is operated upon in the same way. If in the midst of such a data stream different operations need to be performed or other data not included in the stream is required, the pipeline is broken and execution returns to the rate associated with scalar code.

The advent, in 1974-75, of the CDC Star-100 made the advantages of vector computing clear. Unfortunately, it did so in a way which made the disadvantages clear as well. The Star came with an unprecedented million words of magnetic core memory (8 MB), which literally filled a quarter of the building in which it was located. Because this kind of memory was extremely expensive, relatively slow memory was used in order to make this large amount of it affordable. The result was that scalar code ran a factor of 256 slower than vector code on this machine. 256 is a very large number. It was viewed in two ways at the time. Some, like myself, viewed it as the ultimate incentive to vectorize absolutely everything in the numerical algorithm. Others, and there were very many others, viewed it as an incentive to ignore the machine entirely. As a result, there was not a lot of competition for time on the Star, while, to one who knew what to do with it, the Star was the largest and fastest computer in the world. When, in due time, vector computing came to dominate the supercomputing field, early users of this technology on the Star were rewarded immediately with exceptional performance on the new machines.

Even though this account is intended to concentrate on supercomputers I have known, it would be unfair not to mention the other special supercomputer of the mid 1970's, the Illiac IV. This machine was located across the San Francisco Bay at NASA Ames Research Center. Like the Star, it did not have a large user community. It did not exploit vector pipelining to speed execution but relied instead on a very similar concept, that of simultaneous parallel execution of identical instructions by many relatively inexpensive processors. At the time, at least at Livermore, this was thought to be a less flexible approach, but from the vantage point of 1996 this is not so clear. Vector instructions were also limited mainly to performing identical operations on long strings of data values. As we will see, simultaneous parallel execution did not die with the Illiac IV but arose about a decade later in a number of specialized computers, the most popular of which was the Connection Machine.



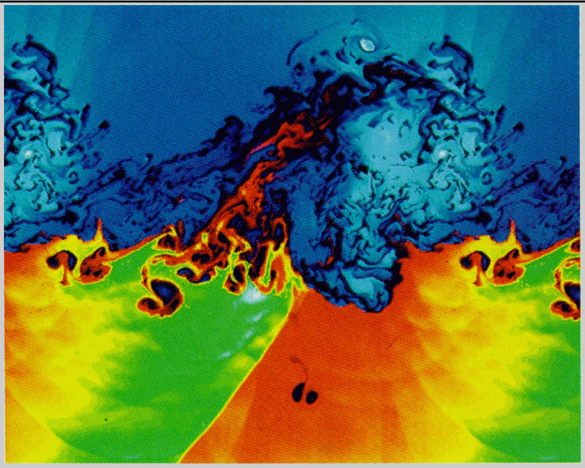
The above figure shows a result of the largest calculation I was able to fit into the memory of the Star-100 at Livermore. This BBC code simulation with 36,000 grid cells, a record in that day, modeled the shock driven implosion of an inter-stellar gas cloud, leading, in the model, to the formation of a star. The density contours and velocity arrows represent state-of-the-art graphics of the time.

Toward the end of the 1970s the machines that brought vector supercomputing into the mainstream appeared. These were the follow-on to the ill-fated Star, the CDC Cyber 205, and the much more popular Cray-1. I was at Livermore at the time, where the Cray-1 was chosen, and therefore I have no experience with the Cyber 205. The Cray-1 had 1 million words of memory (8 MB), like the Star, but this was transistor memory, rather than magnetic core memory. This technological advance brought the ratio of vector to scalar execution speed down from the Star's value of 256 to a range between 5 and 10. Thus the machine did not appear to come to a grinding halt when it encountered a section of scalar code (such as, for example, the operating system). The joy of computing on such a machine, after the Star, provided at least part of the reason for the resistance a decade later to massively parallel SIMD computing, where this critical ratio was brought back to 1024 or more.

The Cray-1 brought to vector computing another innovation, vector registers. The Star relied on the ability of its memory system to bring into its pipelined functional units two operands and a control bit and to bring back to memory one result word every clock cycle. It was therefore a "memory-to-memory" machine which was forced to store all temporary results in main memory. Because vectors had to be so long to be efficient on the Star (ours in the BBC code were the size of the entire grid of 10,000 to 36,000 cells), these temporary results had a way of filling up the whole memory. To attack problems of reasonable size, I was forced to write my code with only one machine instruction per Fortran line, so that the compiler would introduce no temporary vectors. I also had to manage my temporary vectors explicitly; their names, *ws1* to *ws5*, pervaded the text, so that the program appeared to have been encrypted. No one I have met who has written a program like this, and I have met several who have written them recently for massively parallel machines, has ever wanted to repeat the experience.

The vector registers of the Cray-1 saved programmers from the task of personally equivalencing and managing their scratch storage. The bulk of temporary results were created by the compiler and held in registers, each only 64 words long. Scalar variables in vector loops were promoted by the compiler to vector register variables, with no storage space wasted, so that descriptive names could be used for them in the program. Finally, efficient vector performance was achieved on this machine with vectors of only a few hundred elements. A program could designate hundreds of these vectors without taking up much space in the 1 million word memory, and hence they could have descriptive names, and the program could be read easily by humans. There has been a tendency for massively parallel computers of recent years to backslide on these important gains, and there has been a corresponding resistance to these machines from the programming community.

The temporary storage provided by the vector registers in the Cray-1, allowed the bandwidth of the memory system to be reduced over that of the Star, from 3 words per clock to only 1 word per clock, without much impact on overall code performance. This, of course, greatly reduced the cost of the machine (which is not to say that the machine was cheap). This same memory bandwidth of 1 word per clock for each set of floating point functional units (each adder and multiplier combination) seems to have become a sort of industry standard for microprocessors in accessing their cache memories. However, microprocessors do not have the quantity of temporary storage in their registers that the Cray-1 enjoyed, and therefore they do not perform as many floating point operations per clock on most codes, even if those codes are written in vectorized form. Microprocessors have, however, taken an additional step in reducing the cost of the memory system. Unlike the Cray-1, they have cache memories in addition to registers, but this great innovation and its ramifications will be discussed later.


This color image of gas density shows the nonlinear instability of a Mach 2 supersonic slip surface, an instability thought at the time of the calculation not to exist. This calculation was performed with the PPM code on the Cray-2 at the University of Minnesota in 1986, but the pioneering investigation of this phenomenon was done 2 years earlier on the first 2-processor Cray XMP at Livermore in the intervals between debugging of the LTSS operating system, thanks to the generosity of Bob Borchers and of Berni Alder. A 2-D grid of 720x360 cells, regarded as enormous at the time, was used in the first simulations. The calculation shown here used a grid of a million cells. The lower fluid is 10 times denser than that above it, and the many shocks appear like creases in this image. Analytic studies of this instability, motivated by these supercomputer simulations, were later carried out by Andrew Majda and his associates at Princeton, and detailed comparisons with higher resolution PPM simulations were performed by Jeff Pedelty and Gene Bassett in their Ph.D. theses at Minnesota. Images from these later simulations were first animated on Karl-Heinz Winkler's high-speed Gould graphics system at Los Alamos, which introduced the art of interactive computer movies of fluid dynamics calculations. A year later my group duplicated this system at Minnesota.

The introduction of vector computing gave an enormous boost to supercomputing performance. It allowed fast CPUs to run at a speed determined by the memory bandwidth for particularly favored memory access patterns and not to be impeded by the generally high memory latency. However, this performance boost, roughly a factor of 10 from the CDC 7600 to the Cray-1, came at the cost of a restrictive programming model in which large numbers of identical operations had to be performed in sequence. The CDC Star-100 brought in the era of vector computing, but it prepared only a few of us for the machines which followed. The supercomputing community as a whole took several years, beginning with the introduction of the Cray-1, to adjust to vector computation. Already in the early 80s several people could see the next revolution coming, among them, and known to me, George Michael at Livermore and Don Austin at the DoE in Washington. They were determined to make this transition more rapid and more orderly. This next revolution was that of parallel processing, or massively parallel processing, as it came to be known. The mechanism for the orderly passage was the multi-agency High Performance Computing and Communications Initiative.

III. MASSIVELY PARALLEL SUPERCOMPUTING

In the early 1980s, it seemed that every conference included the display of viewgraphs with ascending lines plotted on semi-log paper both documenting and predicting supercomputer performance. The advent of vector supercomputing had apparently kept us on the mandated "historical curve" of exponentially increasing supercomputer performance. The dots on these diagrams through which the historical curve was drawn were rather sparse during the decade of the 70s, and no doubt few in the audience had ever computed on or, much less, even seen the Star-100 and/or Illiac IV. For most people, the historical curve during the 70s consisted of a jump discontinuity toward the end of the decade, when the Cyber 205 and the Cray-1 appeared. In any case, it was said that physical limits of CPU clock speed were being approached, and this appears to have been true. To keep on the curve, surely this community's manifest destiny, would require devoting more CPUs to running a single program.

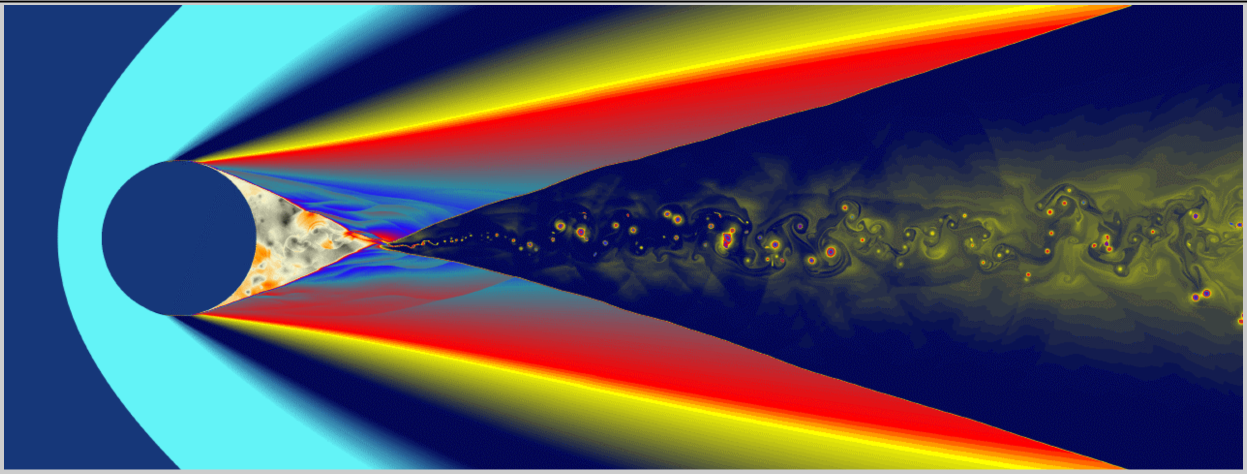
During the 1980s, the number of processors on Cray vector computers increased from 1 to 8, and to 16 a year or two later. However, these expensive machines were all shared resources managed by centers whose interests demanded that aggregate computing throughput be maximized. As a result, the vast majority of jobs run on these machines used only a single processor, and there were few, if any, advantages to using more. Thus individual Cray programmers were able to do more runs, but generally not bigger runs on these machines. This effective stalling of machine performance as perceived by the user helped to drive the transition to massively parallel machines, which originally, like the Star a decade earlier, did not support multiple users well. These machines, unfortunately, brought back many of the disadvantages of Star computing along with this principal advantage. One must remember, however, that, unlike the earlier case of the Star, working with these machines was agreed by nearly all to be pioneering the future of supercomputing, and therefore of a special value to the community. It was generally agreed that speed of light limitations in CPUs would eventually force everyone to move to machines with hundreds or even thousands of processors.

The High Performance Computing and Communications Initiative was the creation of many people concerned that the U. S. Government should resume its former leading role in accelerating the pace of supercomputer system development. This program was established as an unprecedented cooperative effort of several independent federal agencies. The program also explicitly acknowledged the vital role of new numerical algorithms in assuring the success of this venture, and tied this algorithm development to scientific applications of grand scale and wide impact, which became identified as "grand challenges" in computational science. This program was highly successful and served to establish meaningful and lasting working relationships on a national scale between computer designers, system software developers, application code developers, and scientific researchers.

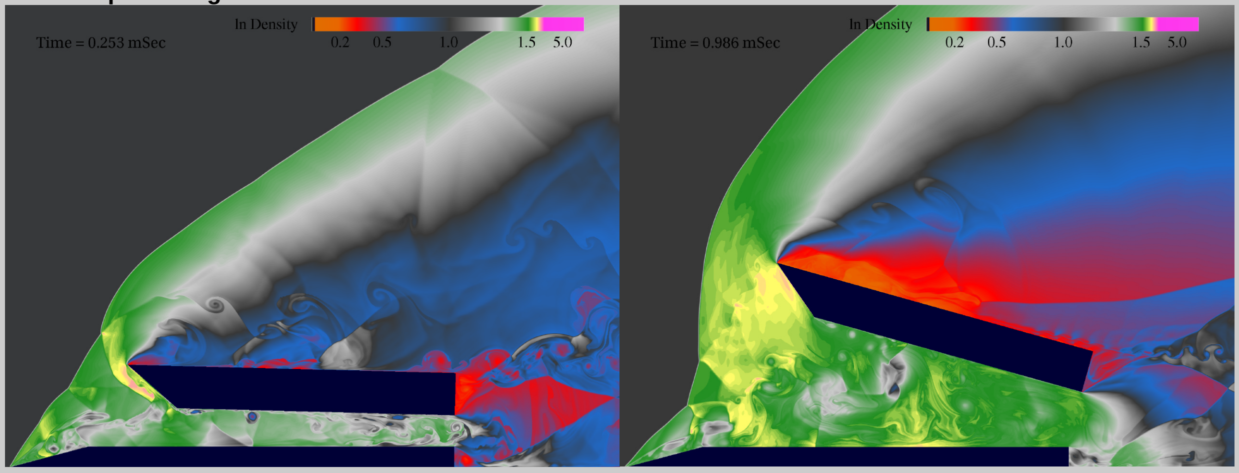
The massively parallel machines, or MPPs, came in two flavors, SIMD (single instruction, multiple data) and MIMD (multiple instruction, multiple data). In principle, these flavors were very different, but in practice their common requirement that hundreds or thousands of processors be kept busy drove a common mode of operation. This mode was essentially SIMD, just like vector processing. If the different processors did anything very different, they would inevitably get out of step. Then either processing loads would have to be readjusted, a very difficult and expensive operation, or many processors would have to wait idly, destroying the chances of the programmer for the multi-Gigaflop/s performance targets demanded by the supercomputer center management. One did not hear very much about the programs that let processors sit idle, but they may nevertheless have existed. My principal experience of these machines was with the Connection Machines CM-200 and CM-5 at the University of Minnesota's Army High Performance Computing Research Center. By this time I was leading a team of people, and, regretfully, they, Steven Anderson, Thomas Varghese, Kevin Edgar, Gene Bassett, and David Porter, got to actually run the programs. I was involved in the structuring of the PPM codes for these machines and the specification, with Matt O'Keefe and Hank Dietz, of a precompiler, written by Terence Parr, for a restricted Fortran dialect, Fortran-P, which eased the writing of high performance code for these machines. This precompiler helped us to keep a common set of Fortran-77 codes, so that we could move rapidly to new machines while at the same time using the older ones. As mentioned earlier, the qualities of a good program for these machines, whether SIMD or MIMD, were quite similar despite the lack of a common vendor-supported programming language.

The Connection Machine was one of a group of machines introduced in the late 80s which brought to supercomputing a new concern not only for the pattern by which data is organized in memory but also for precisely where particular data elements actually reside. The Connection Machine, like most MPPs, had a distributed memory. This meant that each processor had a local memory for which the latency was much lower and the bandwidth much higher than for the rest of the memory of the machine. The data parallel programming model for this machine presented the memory as globally shared, but the machine seemed to come to a halt whenever this property was actually exploited in the program.

Distributed memory machines with low bandwidth, high latency interconnections between the memories of different processors favor algorithms

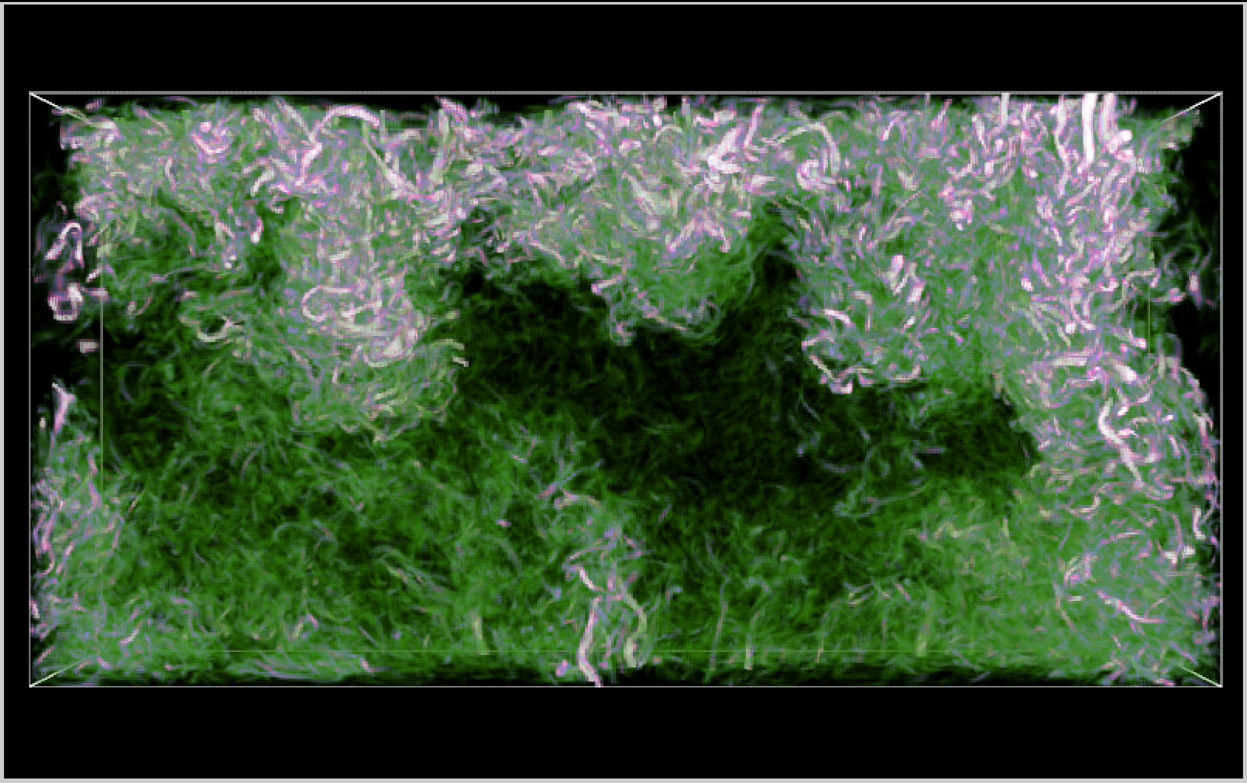


These images of density distributions in 2-D supersonic flow about a cylinder and a 2-D projectile with a movable casing, or sabot, represent experiments with the PPM code exploring the possibility of carrying out regular grid computations, much preferred by both the SIMD Connection Machine and the PPM algorithm, of irregular fluid flow problems. This work with Kevin Edgar and Steven Anderson at Minnesota and with Kurt Fickie at the Army Research Laboratory exploited the 8 Gflop/s speed of the Army High Performance Computing Research Center's 512-node (and 2048-processor) Connection Machine on our PPM code to describe these stationary and moving object boundaries in a very simple fashion on the 4028x2048 grid used for the cylinder and the 2536x2048 grid used for the projectile calculation. The calculations shown use the inviscid Euler equations, while similar computations using the viscous Navier-Stokes equations give similar results but with less fine-scale detail.



which rarely refer to memory locations which processors do not "own." Based upon my earlier experience designing out-of-core PPM simulations for Cray computers, I realized that if a subdomain of the grid to be updated by a particular processor were augmented by a few cells along each boundary, then the grid update could be accomplished completely without reference to data outside the processor's local memory. Thus once each time step, or each 1-D sweep for our PPM algorithm, we could exchange this boundary data with neighboring processors and continue the computation using only local data. During the grid update, neighboring processors would perform some redundant operations on this overlapped data, but the replacement of many small data exchanges by a single, massive data exchange would more than compensate for the lost time from this redundant work.

This trade-off between redundant computation and interprocessor communication is a feature of a wide variety of numerical algorithms. On distributed memory machines it can be critical to good performance, and I believe that it should therefore be possible for the programmer to control this trade-off through the programming language if he or she desires. By working with Woody Lichtenstein of Thinking Machines Corp., our group was able to have this facility introduced into the Connection Machine's Fortran compiler, in release 2.1, but unfortunately Thinking Machines declared bankruptcy only a year later.

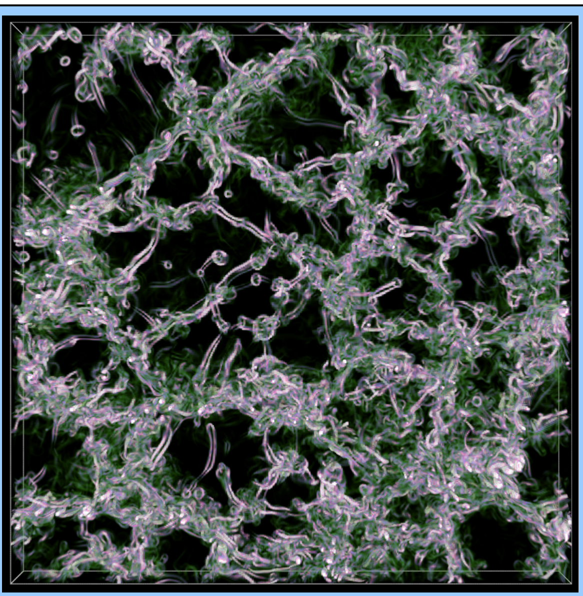


Through the National Science Foundation's Supercomputer Centers Program, established in 1985, U.S. computational scientists were given access to state-of-the-art supercomputing hardware for basic research. As part of the High Performance Computing and Communications (HPCC) Initiative, a set of Grand Challenge Application Groups were set up, including a team to which my group belongs which is centered at the University of Colorado. Also as part of the HPCC program, NSF's Pittsburgh Supercomputer Center was assisted by ARPA in purchasing the first Cray T3D machine. The images above and on the next page are taken from a simulation carried out by David Porter, as part of this team effort, on 256 processors of this machine in "friendly user" mode in its initial months of operation. Our group at Minnesota, together with Silicon Graphics, developed a special, very high resolution visualization system called the PowerWall™ in order to explore the results of high resolution simulations like this and the Connection Machine runs shown earlier. These images were among the first animated on the PowerWall prototype in the Silicon Graphics booth at the Supercomputing '94 exhibit. Through an NSF grant, we built the first production version of the PowerWall at our Laboratory for Computational Science & Engineering at the University of Minnesota. The system uses Ciprico RAID disk arrays to feed image data to a Silicon Graphics Onyx machine driving 4 display panels of a single, large screen at 3200x2400 pixel resolution and 15 images per second.

In 1994, my group, and David Porter in particular, was fortunate to obtain access to the first Cray T3D machine at the NSF's Pittsburgh Supercomputer Center. This machine demonstrated the technical feasibility of interconnecting 512 or even 1024 processors in such a way that the latency of one processor obtaining a word from the memory of any other was as little, reportedly, as 2 or 3 microseconds. The interconnection bandwidths were good as well. However, my team had by this time begun programming so defensively for these distributed memory machines, as described above, that these great technical achievements had no impact on our code's performance. Instead, we were grateful that our codes could be moved easily to the T3D, and we exploited this fact by grinding out simulation after simulation of convection in the outer regions of the sun on this machine. The very special interconnection of processors in the T3D has, I believe, had a lasting influence on supercomputer design. This machine has also, I believe, helped to drive home another important point about microprocessor-based supercomputers, namely the relationship of cache memories to performance. The DEC Alpha microprocessors in the T3D do not have the secondary, off-chip caches which the microprocessor designers intended for them. Our group discovered early on that if we ran the same version of our PPM code on a desktop workstation built around the same DEC Alpha microprocessor as in the T3D, with the same clock rate, we would obtain roughly twice the performance as on the T3D processor. Presumably, this was because the desktop processor had a second level, off-chip cache memory. Our group had encountered the importance of cache memories for microprocessor-based supercomputer performance about a year earlier, and that story, told in the next section, will bring us to the present time, and will lead to a vision for the future as well.

IV. CACHE MEMORY SUPERCOMPUTING

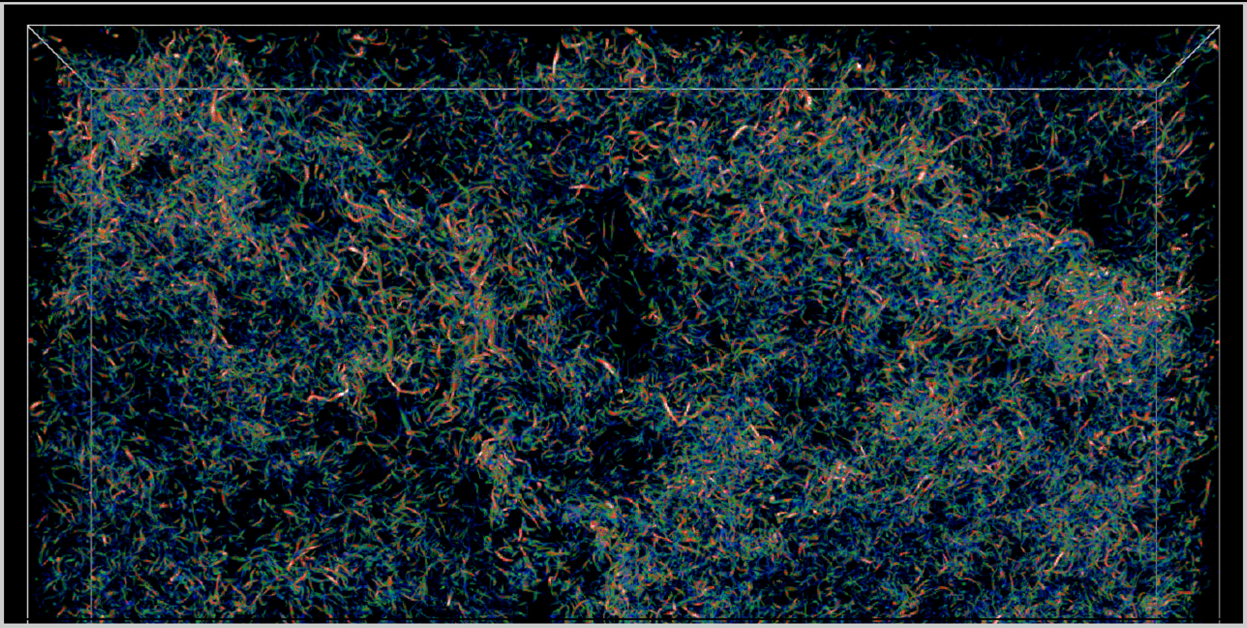
Many features of microprocessor CPUs and their supporting memory systems were first introduced in supercomputers. This is true of multiply-add and conditional move instructions, pipelined functional units, and also of interleaved memory systems. However, cache memories have only entered the supercomputing arena along with the microprocessors which rely upon them, and this has been a very recent development. In spite of using Unix workstations for years for convenient code debugging and for visualization of the results of supercomputer calculations, I did not seriously attempt to extract floating point performance from a cache memory microprocessor until the summer of 1993.


A layer of gas about 20 times denser on the bottom than at the top is used to represent a small chunk of the outer region of the sun. Heat is continually supplied at the bottom of the layer and the top confining surface is kept at a constant cool temperature. The convective instability of this layer has produced turbulent plumes of cool gas that descend from the top boundary and warm upwellings at the bottom surface. A perspective volume rendering technique is used to make visible only regions of the gas that are rapidly spinning or shearing. The side view of a thin slice of this gas layer (shown on the previous page) shows these plumes, while the view of a thin horizontal slice at the top of the layer (shown here) shows the network of thin vortex tubes that delineate the convection cells there.

In 1993, after performing a series of simulations on the Cray-2 and CM-5 at the University of Minnesota which investigated the properties of homogeneous, compressible turbulence, David Porter and I were led to the conclusion that we needed to use a grid of 1024^3, or over a billion cells, in order to get results which were free of the contaminating effects of our periodic boundary conditions, on the largest scales, and of our PPM scheme's numerical dissipation, on the smallest scales. Our study of the convergence properties of these simulated flows as the grid was progressively refined clearly indicated that at the 1024^3 grid resolution we would have the uncontaminated data we sought. As a result, we began to look around for a machine which might be able to perform such a simulation, by a factor of 16 the most ambitious we had by then attempted. Our CM-5 did not have enough memory to hold the data required to describe the fluid state on such a grid. Our Army center submitted a proposal to expand this machine's memory by a factor of 4, at a cost of millions of dollars, but this request was denied. I also requested the month of dedicated time, spread out over a year, which this run would require on our supercomputer center's Cray C-90. I proposed to do the simulation out of core on this machine, since its memory was small but its I/O performance could compensate for this deficiency. This request was denied by our review panel as an inappropriate use of the machine. They felt that this simulation was too demanding, and perhaps they were right.

Later that year, in the midst of our frustration, Dave Perro from Silicon Graphics (SGI) in California began to explore with my group ways in which we could help him to demonstrate the capabilities of SGI's new line of multiprocessor machines, the Challenge XL servers. From this conversation, and from several others with him and with Dale Hanson, our local SGI salesman, over the next month, the idea of doing the billion-zone turbulence simulation on these new machines emerged. These top-of-the-line SGI machines could accommodate up to 2 GB of memory at very reasonable cost, so that, we thought, our simulation would only need 10 machines to hold all the necessary data. Since we were already prepared to do the calculation on our Cray C-90 swapping chunks of the grid, which could be updated independently (as we did on the Connection Machine), back and forth to disk, there should be no problem in dividing the computation between the 10 SGI machines. Dave Perro and Dale Hanson suggested that the necessary machines could be collected in SGI's manufacturing facility during the slow business period immediately following the end of a fiscal year. After a brief period of testing and "burn in" by our team, in which the special power of these machines acting in concert would be demonstrated, these machines, the reliability of their components now thoroughly verified, would be reconfigured and sold to customers.

In order to make this plan work, of course, we would have to get our simulation done quickly, so that customer shipments would not be delayed. My initial reaction to this idea was that we had never seen particularly notable performance on our SGI machines for fluid dynamical applications. Dale urged us to



This image shows a perspective volume rendering of the vorticity distribution in a slab of turbulent fluid representing an eighth of the depth and half of the height of our group's PPM simulation of homogeneous, compressible turbulence on a record-breaking grid of 1024^3 cells. In 1993, despite other attempts, we were only able to do so large a computation on a prototype supercomputer assembled from commercially marketable components by Silicon Graphics in their manufacturing building in Mountain View, California. This Challenge Array prototype, constructed expressly for this computation, consisted of 16 Silicon Graphics Challenge XL servers, each with 20 MIPS R-4400 processors running at 100 MHz and each with 1.75 GB memory, 12 GB disk storage, and an Exabyte 8-mm tape drive. 20 FDDI rings interconnected these machines in a 3-D torus topology. The PPM simulation generated 500 GB of scientific data for later analysis in our lab in Minnesota.

work harder on this, and he noted that up to 32 R-4400 processors running at 150 MHz could be accommodated in each chassis along with the 2 GB memory. Dale got us rapid delivery of the eight 100 MHz R-4400s we had ordered for our graphics lab at the Army center, and we, principally David Porter, Steve Anderson, and myself, began modifying our 3-D PPM hydrodynamics code to see what these processors could do. This turned into a crash course on the peculiar demands of programming for a cache memory and discovery of compiler flags which we had to that point never imagined. Each speedup which we reported to Dale somehow mysteriously caused the machine configuration being discussed with SGI in Mountain View to have fewer processors, less MHz, or both, but enthusiasm on our team was growing to such a degree that leaks of these successes to SGI, which was desperate to keep the cost of this experiment within bounds, could not be avoided. We wanted, of course, 12 fully configured machines with the best processors available and a full month of dedicated time. It should come as no surprise that we got somewhat less than this.

In August and September of 1993, our group, with David Porter and Steve Anderson on the scene, carried out the planned billion-cell PPM simulation of compressible turbulence in the SGI manufacturing building in Mountain View, California. We ended up with 16 machines, with 1.75 GB memories and with 20 R-4400 processors (100 MHz) each. In the end it was the memory requirement and the network topology which determined the number of machines we used. These machines were interconnected with 20 FDDI rings in a 3-D torus topology. Our month of dedicated time turned out to include a week of actually building the machines, a week of getting things running, a week of debugging networking problems and dealing with unanticipated problems like the bizarre phenomenon, peculiar to direct mapped cache machines, of "page coloring," followed by a final, glorious week of computation. We ultimately got the R-4400s going at 40% of their peak performance, and, overall, obtained a sustained 4.9 Gflop/s from the entire system on this enormous problem, the largest fluid dynamics computation to that date, to my knowledge. This was no world speed record at the time, but doing this grand challenge problem on a cluster of workstations, albeit very fancy workstations, was something special. As a result SGI came up with a special name for this new supercomputer, the Challenge Array.

Our group's billion-cell PPM turbulence simulation at SGI signaled, for me, the arrival of a very attractive alternative form of supercomputer, one cobbled together using standard networking technology from a reasonably small number of powerful, cost-effective machines supported by a broad commercial market. The broad commercial support of the components of this supercomputer made it possible to assemble it for this experiment and then to sell off its parts to customers. This concept was, I believe, unprecedented. It signaled a dramatic shift in the scientific computing community, with technological advances of great importance to supercomputing being driven by a commercial market of customers not engaged in this activity. The CDC Star-100 was the result of a direct commission by a DoE supercomputer center, but the SGI Challenge Array was born without a single purchase order.

I should point out that several groups around the country had by that time built networks of standard single-processor workstations and used them to address single problems. About a year earlier, I had, in fact, adapted my PPM code using PVM (Parallel Virtual Machine) message passing to run on Karl-Heinz Winkler's cluster of IBM 560 workstations at Los Alamos. However, the aggregate performance on this sort of system was only comparable to that of a single Cray vector processor. By coordinating the efforts of several multiprocessor workstations, however, we were able to achieve full supercomputer performance, bringing cluster computing into the supercomputing arena.

Our billion-cell turbulence computation on the SGI Challenge Array prototype took advantage of the sheer size of this problem in order to simplify the task of programming an efficient PPM code implementation. I knew at the time that success of this new type of supercomputer would require similarly efficient performance on much smaller problems, including those which can be completed in a day, or even an hour. The subsequent work of figuring out how to get our PPM gas dynamics code going fast for small problems on these machines fell largely to Steve Anderson and myself, with assistance from David Porter and Joe Habermann. The lessons we learned over the next year are quite general and therefore worth noting here.

The principal features of an efficient program for a shared memory multiprocessor (SMP) machine, like today's SGI Power Challenge or DEC AlphaServer 8400, are as follows.

- *Task identification.* The programmer identifies a series of tasks, to be performed in a particular order, along with synchronization conditions. Each task, in effect, consists of bringing a data context into the cache memory, performing a sequence, and hopefully an extensive sequence, of arithmetical operations upon it using the cache as a scratch work space, and moving final results back to main memory.

- *Data access.* The tasks are most efficient if they reference very little data in main memory, keep the entire task computation within the cache work space, and restrict all main memory references to a series of reads or writes of many contiguous memory locations at a time.
- *Task synchronization.* The multiprocessing is most efficient if there are a minimum of synchronization restrictions which can cause processors to wait idly. Making processors wait idly at the ends of parallel loop structures while others complete remaining loop iterations is a common mistake. For small jobs, storing both an old and a new copy of the primary data in main memory often helps to eliminate synchronization restrictions.

In this programming paradigm, the cache memory serves to overcome the limitations of both the main memory latency and bandwidth. The cache can also support random access patterns (which stay within it) at nearly peak performance. Therefore this programming model is well suited to irregular problems, where not every cell of the grid is updated in precisely the same way. The key property of a suitable algorithm is its ability to do significant work in a local scratch space from relatively little globally shared data. It is not necessary, as with vector programming, to perform long strings of identical operations. It is my belief that the set of algorithms which will perform well under these new constraints is much richer than that familiar from two decades of vector supercomputing. Time will tell if this assessment is correct.

Is vector programming still necessary? Four of my group's hydrodynamics code kernels, with different degrees of vectorization, were recently tested by Steven Anderson in our lab and on a DEC AlphaServer 8400 at Los Alamos to measure performance obtained on three popular, state-of-the-art microprocessors. In the table below, the Cray C-90's hardware performance monitor was used to count the floating point operations (flops) involved and also to report its view of the degree of vectorization. Both the reported % of vector code and the C-90 performance should be used to judge the vector or scalar nature of these code kernels. None of these kernels have been optimized for C-90 execution; they have instead been written with cache-based microprocessors in mind. The sPPM and PPM14 kernels should be viewed as very highly vectorized, at least for the target microprocessors, which pipeline some of the loops in PPM14 which the Cray will not vectorize. PPM19 is scalar code with some vector code, and MUSCL16 is entirely scalar.

The table on the next page reports the microprocessor performance measured for the above four code kernels. Note that the superscalar design of the MIPS R-8000 processor, with its software pipelining compiler, favors vector code by roughly a factor of 2, while the MIPS R-10000 and DEC Alpha processors, with only a single floating point adder and multiplier fed by a single control thread, show at most a 50% performance advantage for vector code. The algorithms in the PPM14 and PPM19 kernels are fairly similar, with the PPM19 kernel being both more complex and more accurate, while the sPPM and MUSCL16 kernels are both much simpler and less accurate than these. Comparison of elapsed times for PPM14 and PPM19 gives a rough indication of time-to-solution, and comparison of the flops per cell update gives a rough idea of the unnecessary computation required for a vectorized expression of algorithms of this type. The scalar code PPM19 is favored on both the R-10000 and the DEC Alpha chips. The MUSCL16 kernel has been formulated to use a minimum of space in the cache memory, at the cost of complete obscuration of its original vector form.

Characteristics of Four Gas Dynamics Code Kernels as Measured by the Cray C-90 Hardware Performance Monitor

CODE	Flops per cell	% Vector	sec. per Δt	Mflop/s
sPPM	1647	99.8%	1.19	363
PPM14	2671	86.3%	4.28	164
PPM19	1715	93.3%	3.91	115
MUSCL 16	950	0.1%	15.7	15.9

V. CAN WE PUT ALL THIS TOGETHER?

It is tempting to think of future supercomputer designs as a synthesis of the successful elements of high performance computation described above — vector processing, parallel execution by hundreds or thousands of processors, and shared memory multitasking with cache memory microprocessors. But are all these elements compatible? Clearly, the delivery of one useful floating point result per clock, which we have consistently seen from vector processors on our PPM gas dynamics codes, could be assured by the construction of a microprocessor with a vector architecture, like the Cray-1, working out of a special cache memory. However, our performance tests of the MIPS R-10000 microprocessor, cited above, operating on either scalar or vector code indicate the delivery of from half to three quarters of this performance for 32-bit arithmetic (and from 71% to 76% of this for 64-bit arithmetic). Also, a vectorized expression of our algorithms comes at a cost of about 50% additional computational labor. Hence it is not clear that there is much to be gained from accepting the additional restrictions of the vector programming model if one has already accepted the need to work out of a cache memory. In particular, scalar processors would certainly be favored if it were to turn out that two scalar processors could be purchased for the cost of one vector processor.

Combining shared memory multitasking, as described for a single SMP at the end of the last section, together with massively parallel computation would have substantial benefits in program simplicity and flexibility, and hence in programmer productivity. However, it is not clear that the combination of these two elements is technically feasible. To see if shared memory multitasking can be made compatible with massively parallel processing, we will sketch a design for a supercomputer with a thousand processors using today's high performance SMPs as building blocks and indicate how our PPM gas dynamics algorithm might be implemented on it. If the imagined supercomputer looks like it could indeed be built at an affordable price, and if the imagined PPM implementation looks flexible, programmable in a month or two, and is likely to be efficient, then we will judge this thought experiment successful and judge the application of shared memory multitasking over a thousand processors as possible and even desirable. This exercise will be somewhat artificial, since PPM is a representative of only a single, but rather broad class of applications. Nonetheless, the trade-offs which will emerge between code complexity and machine complexity and cost are likely to be more general than this specific example. It is hoped that the example will serve to bring them into focus.

Let us begin with a single SMP building block. We will use numbers taken from our experience with our lab's SGI Power Challenge machine and our 3-D PPM code designed specifically for use on clusters of such SMPs. In this SMP machine, a single 1.2 GB/s bus is shared by 16 processors, with at most 160 MB/s bus bandwidth available to any single processor at any time. Simplifying the process to some extent, this SMP updates the fluid state variables on a grid of $n^2 \times k$ cells by decomposing this grid into a set of grid pencils, each of $n \times m^2$ or $k \times m^2$ cells. All the required data to update the grid pencil is read from main memory into the cache memory of a processor, the pencil is updated via a computation which accesses no other main memory locations (except a few constants, such as the time step value), and the new pencil of data is written back to main memory. Using the algorithm referred to as PPM14 in the table above, 1336 flops are performed at a rate of 144 Mflop/s in 32-bit precision to update each cell of the grid pencil for this single 1-D sweep of the algorithm. The width, m, of the grid pencil is chosen to be 4 and the 5 fluid state variables for each cell are placed contiguous in memory. This data structure assures that, regardless of the direction of the 1-D sweep, we read or write many contiguous memory locations at a time and thus obtain good bandwidth for the data transfer. Our choice m=4 forces us to choose fairly large values for n and k, namely 64 and 128, respectively, in order to keep

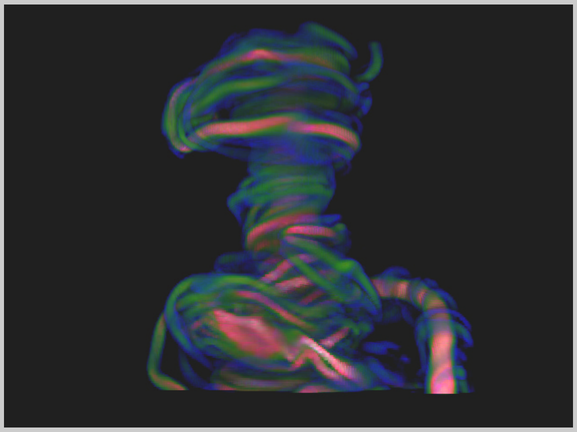
Characteristics of Four Gas Dynamics Code Kernels as Measured on the 90 MHz MIPS R-8000 (4 MB cache), the 190 MHz MIPS R-10000 (1 MB cache), and the 300 MHz DEC Alpha (4 MB cache) updating a 512x512 grid

CODE	*CPU Peak Mflop/s*	*sec. per Δt (Real*8)*	*sec. per Δt (Real*4)*	*Mflop/s (Real*4)*
sPPM	R-8000	6.87	3.89	111
PPM14	(360)	7.32	5.89	119
PPM19		8.47	7.45	60.3
MUSCL 16		4.87	3.84	64.9
sPPM	R-10000	6.00	4.58	94.3
PPM14	(380)	6.65	4.87	144
PPM19		5.68	4.03	112
MUSCL 16		3.68	2.72	91.6
sPPM	DEC Alpha	7.10	5.77	74.8
PPM14	(600)	9.20	7.13	98.2
PPM19		6.52	5.03	89.4
MUSCL 16		4.81	3.96	62.9

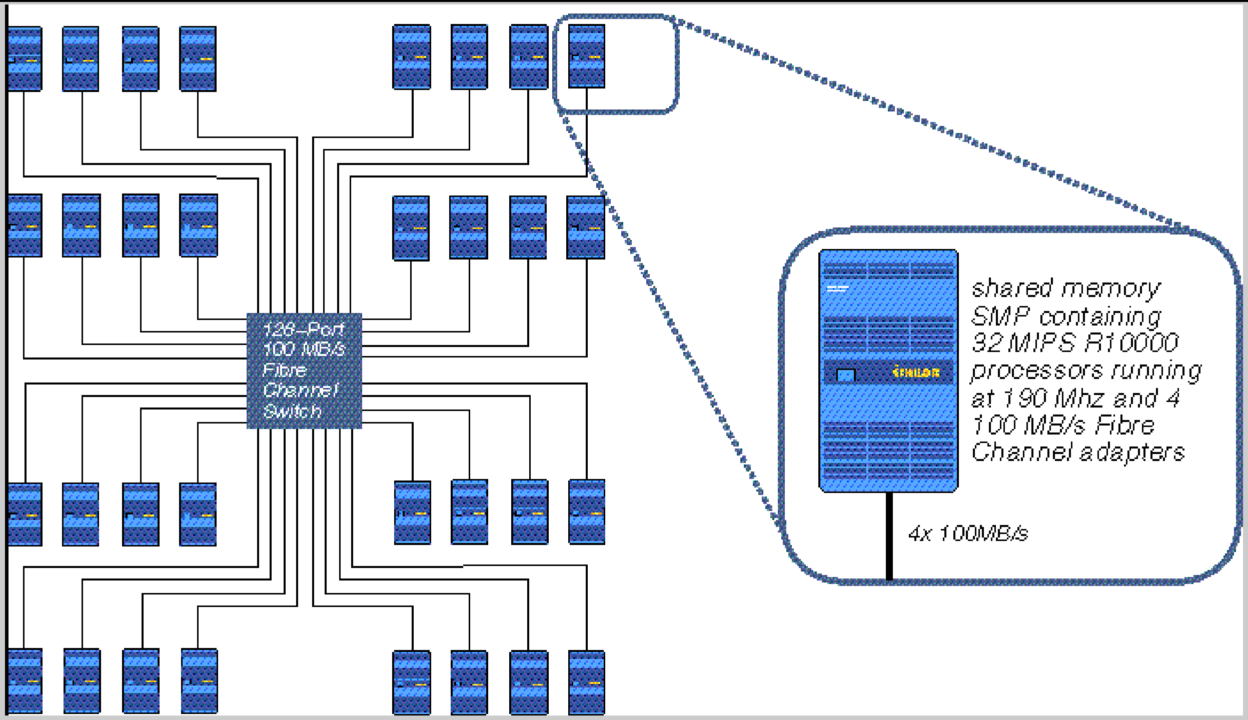
the overall efficiency in updating this grid block as high as 80% (both multitasking costs and costs for boundary computation which is redundant with that in neighboring grid blocks result in this overall efficiency).

We have outlined the manner in which we program an entire 16-processor SMP to update a 3-D grid. If this grid of $n^2 \times k$ cells is one of many subdomains of the grid for our entire problem, then we have described an appropriate task for an SMP in the network of SMPs which will be our supercomputer. After each grid pencil is updated, any necessary boundary data that will be required on the next 1-D sweep to update neighboring subdomains of the grid will be sent to the SMP which “owns” the subdomain now being updated. This receiving SMP will in general be different from the sender, but its location on the network can be determined (it never changes) from the sender’s knowledge of which grid chunk it is working with. If there is a shared memory model supported in the system hardware, in which case this SMP cluster would be called a distributed shared memory (DSM) machine, the dispatch of this message would be expressed as a simple data copy, with array indices communicating the necessary information about the receiver SMP to the system. Otherwise, the message would be sent explicitly, which makes for a somewhat clumsier program implementation but should have the identical effect. We are careful in our PPM program to send this boundary information at the earliest opportunity, so that it will be sure to have arrived when it is needed on the next 1-D sweep. On that next sweep, whichever SMP is updating a neighboring subdomain will prefetch the boundary data it needs from the “owner” of that data, to whom it was sent during the previous sweep. Having specific SMPs own specific subdomains allows both sending and receiving SMPs to exchange messages with each other without either needing to know the identity of the other. This permits subdomains to be assigned dynamically to whichever SMP is free to do additional work in a very simple fashion, just as grid pencils are handed out for processing within each SMP. The result is a simple dynamical load balancing scheme which comes at the cost of doubling the network bandwidth. The asynchronous passing of messages makes the latency of this network nearly irrelevant.

The recent entry of graphics workstation manufacturer Silicon Graphics into the supercomputer arena, particularly after its recent merger with Cray Research, tempts one to consider the possibilities of a supercomputer tightly coupled, either at the same facility or over a fast research network like the NSF’s vBNS, to specialized graphics rendering engines like the Silicon Graphics Reality Engines and high resolution displays like the PowerWall. Such a combination of resources could enable an activity we might call visual supercomputing. Hardware requirements for this are high, but not impossible. At the Laboratory for Computational Science & Engineering at the University of Minnesota, we have explored this possibility with the means at hand, an 18-processor Power Challenge machine acting as the supercomputer running the PPM gas dynamics simulation, a 4-processor Onyx machine with 2 Reality Engines generating the volume-rendered displays with our group’s interactive BoB rendering utility, and the PowerWall providing the high quality display of the results.



An image from this test, carried out by David Porter, showing the vorticity structure of secondary instabilities in a simulated tornado, representative of structures we find in our simulations of convection in the sun, is shown here. A grid of 128×256^2 cells is used to perform this simulation rapidly on this equipment.



The 1024-processor, hierarchically organized machine discussed in the text is indicated in the above diagram. I first began discussing machines designed along these lines with Karl-Heinz Winkler at Los Alamos and Tung Nguyen, then of IBM, Austin, although several features of those imagined designs were different than the one shown above. This diagram is not intended to represent an actual design, as I am not qualified to generate one. Instead, it illustrates the memory bandwidth considerations which are necessary to support my 3-D PPM algorithms when implemented using either the explicit message passing or distributed shared memory multitasking programming models discussed in the text. Distributed shared memory machine designs have recently been promoted by John Hennessy at Stanford, and machines of this type are either offered or planned by more than one supercomputer vendor. SMP cluster computing is the focus of the "Blue" platform option of the Department of Energy's Accelerated Strategic Computing Initiative (ASCI).

It should be noted that this scheme for multitasking over our SMP cluster violates the popular "owner computes rule" of data parallel programming for MPPs like the Connection Machine. The "owner" of a subdomain acts as the intermediary for the messages relating to that subdomain, but it is not constrained to update it. This flexibility is the key element in straightforward load balancing for dynamically irregular problems. To make this work, we must provide enough network bandwidth so that, periodically, complete subdomain data, not just boundary data, can be sent back to the subdomain owner so that this subdomain can be reassigned, if necessary, to a different SMP for updating as a means of rebalancing the computing loads. It would be delightful for the programmer if this reassignment interval could be so short that no message passing during this interval would be required. If we could have the subdomain task be only updating the grid for a single 1-D sweep or for a single time step, this would be possible. However, this would require enormous network bandwidth. It would also be unnecessary in order to keep computing loads in balance. We have argued above that to keep each SMP task 80% efficient we must choose a subdomain as large as 128×64^2 cells. Computing loads within such tasks are likely to change as the result of the propagation of fronts like shocks, combustion fronts, or multifluid interfaces. These cannot move, in an accurate computation, more than one cell per time step. Thus we would not expect the load balance to change by more than 10% in 8 time steps, and it would almost surely change considerably less in this interval. Thus we are driven to accept the necessity of intertask communication, but by introducing the owner SMPs or, equivalently, a global shared memory as the intermediary for this communication we are able to retain a fairly simple programming model while keeping the network cost within reach. We should also point out that having multiple subdomain tasks reside in the local memory of an SMP for several time steps, in addition to the globally shared data stored there, is a side effect of this need to reduce network bandwidth requirements. Unfortunately, this side effect roughly doubles the memory requirement for the simulation. However, at present it seems that extra memory comes more cheaply than extra bandwidth, so this is a favorable trade.

In order to get a feel for the combination of machine and simulation we are discussing, a little quantitative information is helpful. On an SMP with 16 MIPS R-10000 processors running at 190 MHz, our 3-D PPM code would update a 128×64^2 chunk of grid for 8 time steps at 1.84 Gflop/s (80% parallel efficiency) in 9.12 sec. To keep computing loads well balanced, we must have several such tasks for each SMP in the system. With 16 such tasks per SMP we will estimate that 94% load balancing efficiency will be achieved. Each set of 8 time steps would then require 155 sec. At the beginning and end of each set of 8 time steps, that is, roughly every 3 minutes, all the globally shared data would be redistributed over the SMPs in order to rebalance the loads. Allowing 10% of the time for this process, the time consumed by 8 time steps would grow to 172 sec. With 64 SMPs, possibly organized in 8 clusters of 8 or in 4 clusters of 16, we would have 1024 processors and an aggregate sustained performance of 99.6 Gflop/s for our PPM code on this problem. The grid would total 512×1024^2 cells, and to compute for 10,000 time steps (about 5 wave crossing times) would require 2.5 days. Regular problems with perfect load balancing could be 10% or 20% more efficient, even for grids with 16 times fewer cells and 32 times less running time. Even allowing for shared and private copies of the data for the large problem, the memory consumed on each SMP would be only about 320 MB. The bandwidth of the network link to each SMP required by the efficiencies stated above is enough to transfer 12.5 MB in half the time of a 1-D sweep for one subdomain or to transfer 20 MB in 8 tenths of the 1-D sweep time. Each SMP must therefore be connected to the network at 66 MB/s, the speed delivered by a standard HiPPI channel. Connecting 64 SMPs in a network with links of this speed is somewhat challenging today, but still feasible with off-the-shelf networking equipment. For example, each SMP could have 4 HiPPI interfaces, with each connected to a separate 16x16 HiPPI crossbar switch.

Performance of the Directionally Split 2-D PPM Template Code on MIPS R-8000 CPU's @ 90 Mhz Updating a 1024x1024 Grid

# CPUs	CPUs per SMP	Netwrk Link (MB/s)	Speed-Up	Effi-ciency
1	1		1	100%
2	2	1200	1.85	92%
4	4	1200	3.44	86%
7	7	1200	6.16	88%
14	7	100	11.94	85%
14	7	< 0.5	6.69	48%

Transcontinental Distributed Computing: With perfectly balanced loads representing equal subdomains of a single, large, 2-D gas dynamics problem run by our PPM code, the techniques mentioned in the text of overlapping communication and computation serve to enable two 7-processor SMPs, one at the University of Minnesota and one at NCSA in Illinois, to carry out a fluid dynamics simulation of the merger of two compressible eddies in 2-D nearly as efficiently as if both machines were located in the same room. This experiment was performed by Steve Anderson late at night over the Internet, while NSF's new vBNS research network will provide the bandwidth necessary to effectively unite far greater computational resources over interstate distances for more demanding computational problems. Performance results for this distributed computing experiment on this tightly coupled problem are given in the tables on this and the following page:

Performance of the Directionally Split 2-D PPM Template Code on MIPS R-8000 CPU's @ 90 Mhz Updating a 2048x2048 Grid

# CPUs	CPUs per SMP	Netwrk Link (MB/s)	Speed-Up	Efficiency
1	1		1	100%
2	2	1200	2.26	113%
4	4	1200	4.38	109%
7	7	1200	7.56	108%
14	7	100	14.9	106%
14	7	< 0.5	13.0	93%

In constructing this example, we have used familiar elements – processors, bus-based SMPs, and HiPPI switches – which could be purchased today. By combining these elements we see that 100 Gflop/s sustained performance can be achieved without resorting to an extreme configuration. The Department of Energy's Accelerated Strategic Computing Initiative (ASCI) has a "Blue" platform option based upon clusters of shared memory multiprocessors (SMPs) with a requirement of one teraflop/s sustained performance (with the sPPM benchmark code) in 1998. This exciting program will therefore drive the supercomputer industry to stretch a little, which is the point of the "A" in ASCI after all.

If the hardware and system software of the SMP cluster we have been considering is enhanced so that a globally shared memory with cache coherency is supported, the programming task clearly becomes much simpler, with message passing replaced in the code by simple data copying between globally shared data and data private to an SMP. Such a machine is called a distributed shared memory (DSM) machine, a concept promoted by John Hennessy in recent years. A hierarchy of memory systems, both in latency and bandwidth, is an essential feature which keeps the cost of such a system within reach. This machine architecture offers the possibility of combining massively parallel computing with the shared memory multitasking which allows convenient load balancing in irregular problems. Like all supercomputing systems before it, however, this sort of machine will strongly favor certain numerical algorithms and force others to execute at much slower speeds. The usefulness of the favored algorithms and the ease with which they can be implemented, together with the Gflop/s or Tflop/s these algorithms achieve, will determine the scientific output of these machines. Ultimately it is this scientific output which is the true measure of a supercomputer.

VI. REFERENCES

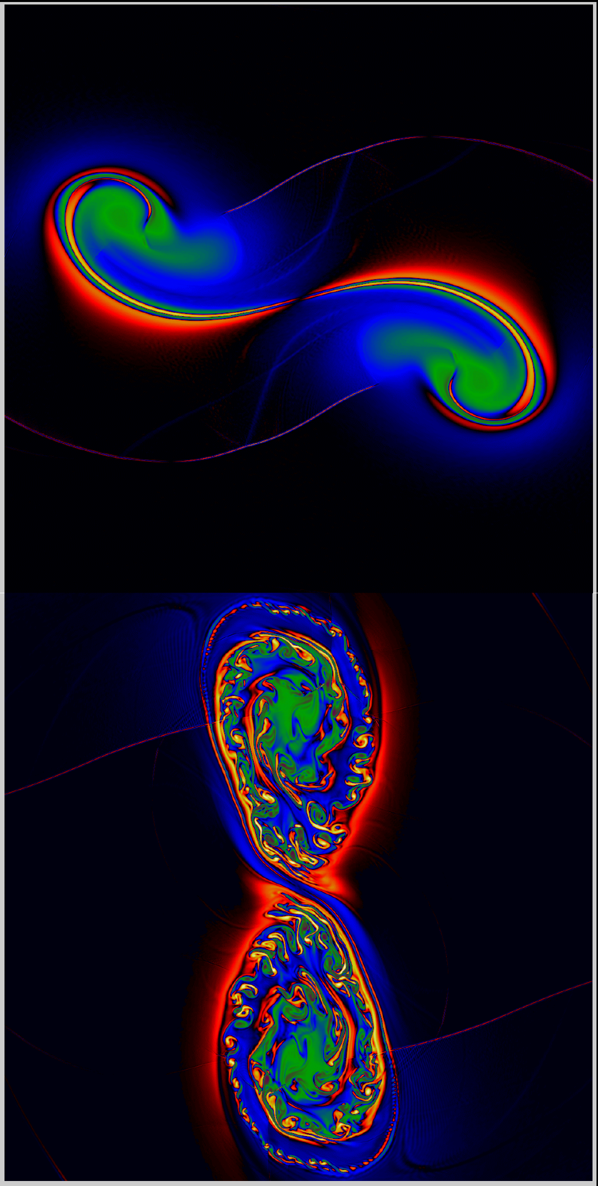
1. P. R. Woodward, D. H. Porter, B. K. Edgar, S. E. Anderson, and G. Bassett, "Parallel Computation of Turbulent Fluid Flow," *Comp. Appl. Math.*, Vol. **14**, no. 1, pp. 97-105 (1995).
2. P. R. Woodward, "Interactive Scientific Visualization of Fluid Flow," *IEEE Computer*, Vol. **26**, No. 10, pp. 13+-25 (October, 1993).
3. W. J. Kaufmann and L. Smarr, *Supercomputing and the Transformation of Science*, Scientific American Library, HPHLP: New York, 1993.
4. S. Karin and N. P. Smith, *The Supercomputer Era*, Harcourt Brace Jovnovich: San Diego, 1987.
5. E. J. Pitcher, ed., *Science and Engineering on Supercomputers*, Springer Verlag: Berlin, 1990.
6. R. B. Wilhelmson, ed., *High Speed Computing, Scientific Applications and Algorithm Design*, Univ. of Illinois Press: Urbana, 1988.
7. *Science*, Special Issue, "Computers '95: Fluid Dynamics," Vol. **269**, No. 5229, Sept. 8, 1995.
8. J. Kuskin, et al., "The Stanford FLASH Multiprocessor," Proc. 21st Intnatl. Symp. on Computer Architecture, Chicago, Illinois, April, 1994, pp. 302-313.

9. "High Performance Computing & Communications: Toward a National Information Infrastructure," a report by the Committee on Physical, Mathematical, and Engineering Sciences, Federal Coordinating Council for Science, Engineering, and Technology, Office of Science and Technology Policy, 1994.

VII. ABOUT THE AUTHOR

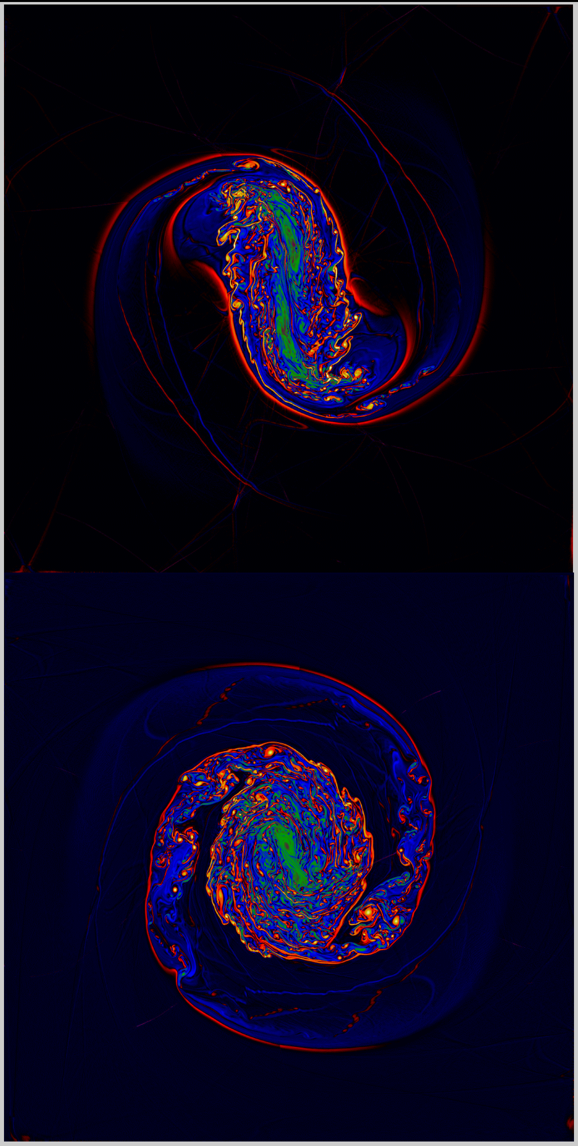
Paul R. Woodward began his supercomputing career as a computational physicist at the Lawrence Livermore National Laboratory in 1968. He received his Ph.D. degree in physics at the University of California, Berkeley, in 1973, based upon a computational project carried out at Livermore. He spent two years at Leiden Observatory in the Netherlands working with Bram van Leer on the development of the MUSCL code for compressible fluid dynamics. Returning to Livermore in 1978, and working with Phillip Colella there, he extended and enhanced the MUSCL scheme to produce the hydrodynamics scheme PPM (the Piecewise-Parabolic Method). In 1985 he joined the astronomy faculty of the University of Minnesota as a Fellow of the Minnesota Supercomputer Institute. In 1991 he became Director of Graphics and Visualization for the University of Minnesota's new Army High Performance Computing Research Center. In 1995 he established the Laboratory for Computational Science & Engineering (LCSE) at the University of Minnesota, which he directs. The LCSE plays an active role in the NSF national supercomputing program through its MetaCenter Regional Alliance with NCSA at the University of Illinois. Woodward's research group at the LCSE is also actively involved in the DoE's new ASCI (Accelerated Strategic Computing Initiative) program, with a special focus on the ASCI "Blue" project involving clusters of shared memory multiprocessors. In 1995 Woodward was presented the Sidney Fernbach Award in large-scale computing by the IEEE.

The work presented here has been supported by the Department of Energy, through the Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and grants DE-FG02-87ER25035 and DE-FG02-94ER25207, by the National Science Foundation, through grants AST-8611404, ASC-9217394, CDA-950297, ASC-9523480, and generous allocations of computer time at the Pittsburgh Supercomputer Center, Cornell Theory Center, and NCSA, by the Army Research Office, through its grant to the AHPCRC at the University of Minnesota, by the Defense HPC Modernization Program, through the Army Research Laboratory, by NASA, through grant



In November of 1995, Kevin Edgar and Steven Anderson simulated transcontinental distributed computing conditions using a cluster of six 16-processor Power Challenge machines at Silicon Graphics' home office. They used a latency tolerant version of the PPM code to simulate the merger in 2D of two eddies, each spinning at half the speed of sound. Four snap shots of the vorticity distribution of the merging eddies are shown here and on the next page. This very finely resolved 2D flow provides an example of the kind of calculations which will be enabled by interconnecting computing resources over fast transcontinental networks like the NSF's new vBNS.

USRA/5555-23/NASA, and from the University of Minnesota, through its Minnesota Supercomputer Institute. Industrial support from Silicon Graphics was critical in performing the billion-cell turbulence calculation described in the text and in developing the PowerWall prototype demonstration. Industrial support to the LCSE from Silicon Graphics, Seagate Technology, Ciprico, Micro Technologies Inc., Ampex, DEC networking division, GeneSys, and Prisa is gratefully acknowledged.



The final two images from the sequence on the previous page showing the merger of two eddies spinning at half the speed of sound.