NODES, MODES AND FLOW CODES

Massively parallel supercomputers seem the best hope for achieving progress on 'grand challenge' problems such as understanding high-Reynolds-number turbulent flows.

George Em Karniadakis and Steven A. Orszag

Understanding turbulent flows is a "grand challenge"[1] comparable to other prominent scientific problems such as the large-scale structure of the universe and the nature of subatomic particles. In contrast to many of the other grand challenges, progress on the basic theory of turbulence translates nearly immediately into a wide range of engineering applications and technological advances that affect many aspects of everyday life.

Numerical prediction of fluid flows is at the heart of understanding and modeling turbulence. However, such computational fluid dynamics simulations challenge the capabilities of both algorithms and the fastest available supercomputers. In 1970 Howard Emmons[2] reviewed the possibilities for numerical modeling of fluid dynamics and concluded: "The problem of turbulent flows is still the big holdout. This straightforward calculation of turbulent flows—necessarily three-dimensional and nonsteady—requires a number of numerical operations too great for the foreseeable future." However, within a year of the publication of his article, the field of direct numerical simulation (DNS) of turbulence was initiated with the achievement of accurate simulations of wind-tunnel flows at moderate Reynolds numbers.[3] (The Reynolds number, a dimensionless measure of the degree of nonlinearity of a flow, is defined as $R = v_{\text{rms}} L/\nu$, where v_{rms} is the rms velocity, ν is the kinematic viscosity of the fluid, and L is a typical length scale at which the energy maintaining the flow is input. At sufficiently high Reynolds numbers, flows become turbulent.) In the last 20 years, the field of turbulence simulation has developed in two directions. First, turbulence simulations are now regularly performed in simple geometries, and extensive databases of flow fields have been constructed for the analysis of turbulent and even laminar–turbulent transitional interactions.[4] Second, simulations of turbulent flows in prototype complex geometries are now emerging.[5] (See figure 1 and the cover of this issue.)

George Karniadakis is an assistant professor of mechanical and aerospace engineering at Princeton University.
Steven Orszag is Forrest G. Hamrick '31 Professor of Engineering at Princeton.

Incompressible fluid flows are governed by the Navier–Stokes equations,

$$\frac{\partial \mathbf{v}}{\partial t} = \mathbf{v}\times\boldsymbol{\omega} - \nabla\Pi + \nu\,\nabla^2\,\mathbf{v} \tag{1}$$

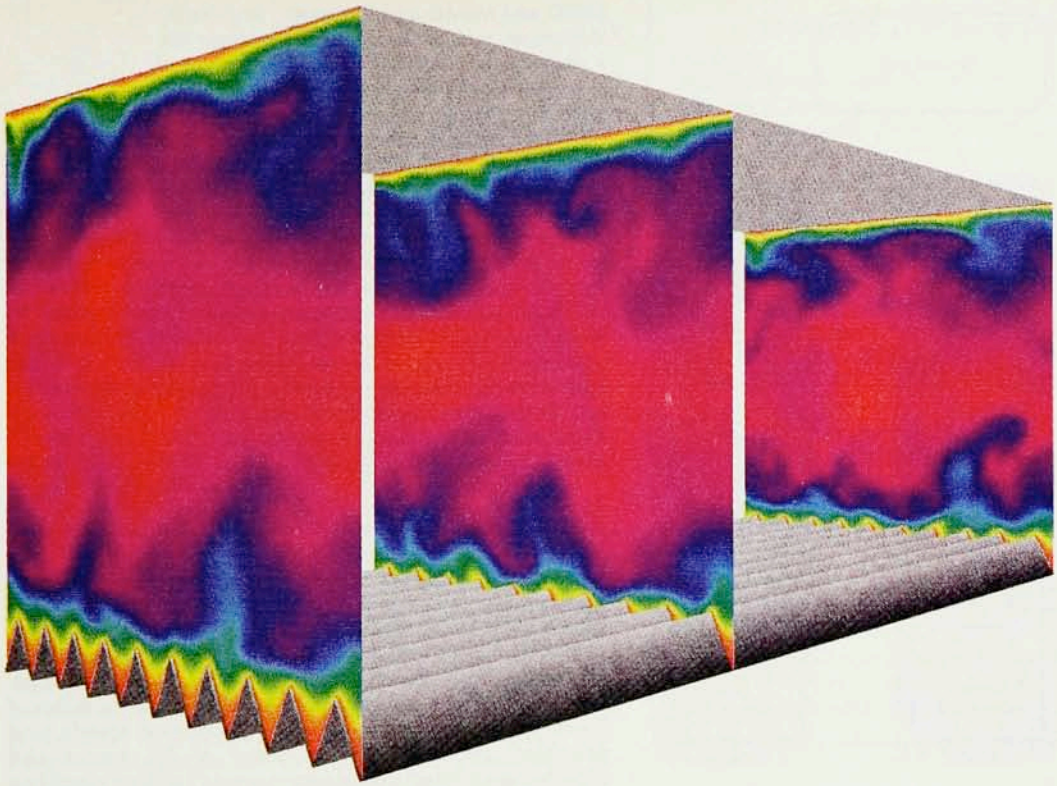
$$\nabla\cdot\mathbf{v} = 0 \tag{2}$$

where $\mathbf{v}$ is the velocity field, $\boldsymbol{\omega} = \nabla\times\mathbf{v}$ is the vorticity field, and $\Pi = p + \frac{1}{2}v^2$ is the pressure head, where p is the pressure. In direct numerical simulation, the Navier–Stokes equations are solved at all scales for which there is appreciable kinetic energy. At large Reynolds numbers, the Kolmogorov theory of small scales in turbulence shows that eddies are appreciably excited at scales ranging in size from L, at which energy input takes place, down to $\eta = L/R^{3/4}$, at which viscous dissipation becomes significant. (See the article by Uriel Frisch and Orszag in PHYSICS TODAY, January 1990, page 24.) Since turbulent flows are necessarily time dependent and three-dimensional and since each excited eddy requires at least one grid point (or mode) to describe it, as R increases, the spatial resolution, or number of modes, required to describe the flow increases at least as fast as $(R^{3/4})^3$.

With conventional DNS methods, the time step must be no larger than η/v_{rms} in order to resolve the motion of small eddies as they are swept around by large ones with rms velocity v_{rms}. Because large-scale turbulence evolves on a time scale of order L/v_{rms}, on the order of $R^{3/4}$ time steps are required. Thus the computational work requirement (embodied in the number of modes times the number of time steps) for DNS of turbulence scales roughly as R^3 and increases by an order of magnitude if R is doubled. This type of rapid increase in resolution and corresponding increase in computational work requirements is the challenge of DNS at high Reynolds numbers and necessitates the use of theory to remove degrees of freedom and simplify the computations.

Two alternative approaches aim to alleviate the computational requirements of DNS of turbulence: Large-eddy simulations[6] use a fixed spatial resolution, and the effects of eddies that are not resolved are modeled using gradient transport ideas such as eddy viscosity. (See the article by Frisch and Orszag.) Reynolds-averaged Navier–Stokes simulations model all turbulent fluctuations theo-

© 1993 American Institute of Physics



Effects of riblets on turbulence as simulated by a spectral-element method on the Delta Touchstone computer. Colors indicate the instantaneous magnitude of the streamwise component of the velocity; the highest values occur in the middle of the channel. Values are shown at three different cross-flow planes. The mean flow is from left to right, and the turbulence is fully developed and statistically steady at a Reynolds number (based on flow rate) of 3500. Computed turbulence intensities indicate that the reduction of fluctuations near the wall with riblets (bottom) results in a 6% percent drag reduction in this geometry. (Courtesy of Douglas Chu, Catherine H. Crawford and Ronald D. Henderson, Princeton University.) **Figure 1**

retically or empirically—not just the ones smaller than the grid spacing. Recently we have studied a variant of Reynolds-averaged Navier–Stokes modeling called very-large-eddy simulation, which has some features of large-eddy simulation: All statistically isotropic eddies are modeled, while large-scale anisotropic eddies are simulated explicitly.[7]

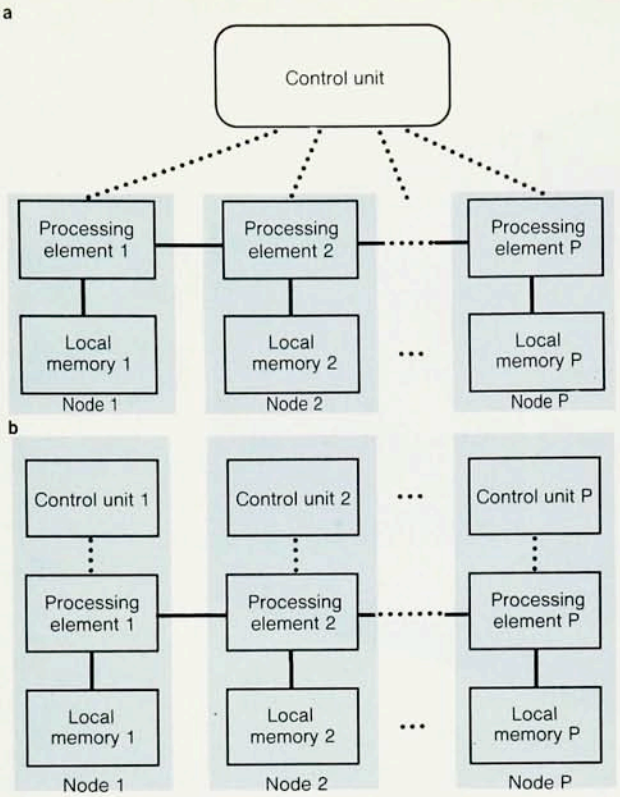
The four images on the cover of this issue illustrate the effect of increasing Reynolds number on flow past a sphere. The top three images, at $R = 300$ (top image), 500 and 1000, are direct numerical simulations. The bottom image, at $R = 20\,000$, is a large-eddy simulation. Each image shows the surface at which the axial velocity is 90% of the free stream velocity, colored according to the local vorticity magnitude. Red indicates high vorticity; white, low vorticity. These simulations were performed on an Intel iPSC/860 32-node hypercube using a parallel spectral-element Fourier code, as discussed later. The large-scale flow pattern is present at all these Reynolds numbers, but for $R \gtrsim 1000$ the excitation of small scales (indicated by vorticity) increases rapidly, making DNS impractical at current capabilities.

The need for parallel processing

There is now a broad consensus that major discoveries in key applications of turbulent flows would be within grasp if computers 1000 times faster than today's conventional supercomputers were available, assuming equal progress in algorithms and software to exploit that computer power and effective visualization techniques to use the results of the computations. This consensus has been realized in the High Performance Computing and Communications Initiative, whose goal is the development and application of teraflop (10^{12} floating-point operations per second) computers in the second half of the 1990s. This thousandfold improvement in useful computing capability will be accompanied by a hundredfold improvement in available computer networking capability.

It is estimated that a teraflop computer could perform Reynolds-averaged Navier–Stokes calculations of flow past a complete aircraft, large-eddy simulation of flow past a wing and DNS of flow past an airfoil, all at moderate Reynolds number (R on the order of 10^8). Following Andrei Kolmogorov's scaling arguments, similar estimates show that DNS of a complete aircraft will require at least an exaflop (10^{18} flops) computer.[8] This example of computing flow past an aircraft is typical: Even with teraflop computing power, progress on real engineering applications will require synergism among computing, theory (to describe the effects of small-scale motions) and prototype experiments[9] (to elucidate fundamental physical phenomena).

We will be able to achieve teraflop speeds in this decade only by using massively parallel supercomputer



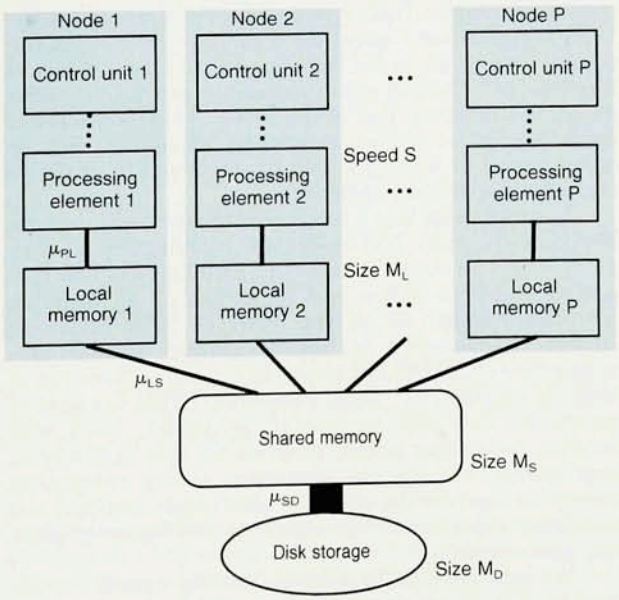
SIMD and MIMD architectures. In a single-instruction, multiple-data-stream computer (**a**), a single control unit manages the operations of P processing elements, each with a local memory. Each processing element performs precisely the same operation or stays idle during each clock period of the computer. In a multiple-instruction, multiple-data-stream computer (**b**), independent control units manage the operations of the processing elements. Each processing element may execute asynchronously from the others; the computations are synchronized at various times by sending messages between processing elements. In either architecture, interconnections between processing elements can be local (mesh topology) or global (hypercube, nonblocking switch, shared memory and so on). **Figure 2**

architectures. This development will fit the pattern of major changes in computer technology and architecture that have occurred about every 20 years since the 1940s. Early computers, motivated by the needs of World War II, were sequential (von Neumann) machines in which one set of arithmetic operations had to be completed before further operations could be executed. By the early 1960s technological improvements in speeds and densities of electronic components led to an increase in computer speeds and memory sizes by about a factor of 1000 over the early prototype computers. In the 1960s sequential supercomputers gave way to supercomputers with pipelined architectures, such as the Control Data Corp 6600 and 7600 computers. Then in the late 1970s, vector supercomputers such as those of Cray Research Inc were introduced, in which vector operations on 64 elements were combined with pipelined operations. By the mid-1980s computer speeds had increased from their mid-1960s values by another factor of roughly 1000 (a factor of 2 every two years) as a result of component speeds (and densities) increasing by a factor of 25, vector and pipelining architectural improvements yielding roughly a factor-of-10 increase, and the first parallel application of several processors to the same job yielding an efficiency factor of nearly 4 on the Cray XMP and Cray 2. We can now foresee a further thousandfold speed increase by the end of the 1990s due to additional increases in parallel efficiencies in excess of a factor of 100 and component speed and density improvements of roughly a factor of 10. It is now expected that teraflop speeds will be achieved by massively parallel supercomputers with a few thousand processors each achieving a maximum speed of approximately 1 gigaflop (10^9 flops).

A medium-size scientific computation that now takes 5 hours on a Cray YMP running at 200 megaflops and that would take over 28 years on a Macintosh at 0.004 megaflops would require less than 4 seconds on the teraflop computer. Similarly, the solution of the grand challenge turbulence problems discussed here that should require 2 weeks per run on a teraflop computer would have required several centuries to run on the Cray YMP and millennia on the Macintosh.

Nodes: Parallel computers

An extensive body of literature on the design and application of parallel computer systems already exists[10] and emphasizes programming models and the parallel efficiencies attainable by them. Here we also wish to emphasize other considerations that determine the effectiveness of such systems for turbulence simulation.



Prototype Parallel Computer can be used to model how memory size, processing speed and data transfer rates (μ_{PL}, μ_{LS} and μ_{SD}) must be matched for efficient parallel computation. A shared memory serves as the interconnect topology among the processing elements, and a fast disk serves as a large data bank. **Figure 3**

We believe that one should approach the design of a computer system to solve physical problems much as one approaches the design of a laboratory to perform an experiment. One must take into account all resolution and computational requirements, including the balance among memory size, processing speed and the bandwidths of various components. However, it is nearly impossible to address these issues in a generic way because of the large variety of existing computer architectures. Here we will try to make some progress by first addressing the issues of programming model and parallel efficiency, and then, in order to address other issues, focusing on the "Prototype Parallel Computer," a system that has many components in common with existing and proposed parallel computers.

A popular taxonomy for parallel computers, introduced by Michael Flynn, divides the programming models into two classes: single instruction, multiple data stream (SIMD) and multiple instruction, multiple data stream (MIMD). In an SIMD computer, such as the Thinking Machines CM-2 or an NCUBE Inc computer, each processor performs the same arithmetic operation (or stays idle) during each computer clock cycle, as controlled by a central control unit. (See figure 2a.) Programs in this model, also referred to as data parallel programs,[11] use high-level languages (for example, parallel extensions of FORTRAN and C), and computation and communication among processors is synchronized automatically at every clock period.

On a multiple-instruction, multiple-data-stream computer (see figure 2b) each of the parallel processing units executes operations independently of the others, subject to synchronization by the passing of messages among processors at specified time intervals. The parallel data distribution and the message-passing are both under user control. Examples of MIMD systems include the Intel Gamma, the Delta Touchstone computers and, with fewer but more powerful processors, the Cray C-90. (See the box on this page for a prescient 1922 description of an MIMD computer.)

While it is often easier to design compilers and programs for SIMD multiprocessors because of the uniformity among processors, such systems may be subject to great computational inefficiencies because of their inflexibility at stages of a computation in which there are relatively few identical operations. There has been a natural evolution of multiprocessor systems toward the more flexible MIMD models, especially the merged-programming model, in which there is a single program (perhaps executing distinct instructions) on each node. The merged-programming model is a hybrid between the data parallel model and the message-passing model and is exemplified in the newest Connection Machine, the CM-5. In this single-program, multiple-data model, data parallel programs can enable or disable the message-passing mode. Thus one can take advantage of the best features of both models.

There is no universal yardstick with which to measure performance of computer systems, and the use of a single number, such as the peak performance quoted by the manufacturer, to characterize performance is often misleading. So that different aspects of the computer system are measured, performance is commonly evaluated in terms of benchmark runs consisting of small code segments ("kernels") and prototype applications. This approach, however, is still dependent on the quality of software rather than just on hardware characteristics. The computer science community has recognized the controversy over performance evaluation methods and has made several recent attempts to provide more objective performance metrics for parallel computers.

Gene Amdahl noticed long ago that the efficiency of a parallel computer system depends critically on the fraction m of the total number of arithmetic operations that can be done in parallel.[12] Consider a computation that requires time T on a single processor. If there are P such processors executing in parallel, the parallelizable operations require time mT/P, while the remaining fraction $(1-m)$ of computations done on a single processor requires time $(1-m)T$. Thus the total time is reduced to $[(1-m)+m/P]T$, giving a scalar performance measure

$$\xi = \frac{1}{(1-m)+m/P} \tag{3}$$

which is the effective number of processors used. For example, if $m=1$, then $\xi=P$, implying that all the processors are used effectively; if $m=0$, then $\xi=1$. Equation 3, called Amdahl's law, shows that massively parallel computers with large P require massively parallelizable computations. For example, if P is large and $m=1-1/P$, then ξ is approximately $P/2$: Only half of the computer is used effectively. The effective performance of the system can be measured by the parallel efficiency $E_P=\xi/P$, which is about $1/(k+1)$ when $m=1-k/P$.

The scalar performance measure ξ can sometimes be misleading, since it may favor inefficient but highly parallelizable algorithms over more efficient algorithms that may be more difficult to map onto a parallel multiprocessor computer.[13] There are several industry-standard benchmark programs such as Whetstone, Dhrystone and Linpack that are for nonparallel systems but have parallel extensions. While these benchmarks have been used extensively in all advanced computer system

'A Myriad Computers at Work'

In his landmark treatise *Weather Prediction by Numerical Process* (Cambridge University Press, 1922), the British meteorologist Lewis Fry Richardson demonstrated remarkable prescience in his description of a futuristic multiple-instruction, multiple-data-stream parallel computing facility for weather forecasting, albeit with human "computers":

"Imagine a large hall like a theatre, except that the circles and galleries go right round through the space usually occupied by the stage. The walls of this chamber are painted to form a map of the globe. . . . A myriad computers are at work upon the weather of the part of the map where each sits, but each computer attends only to one equation or part of an equation. The work of each region is coordinated by an official of higher rank. . . . From the floor of the pit a tall pillar rises to half the height of the hall. It carries a large pulpit on its top. In this sits the man in charge of the whole theatre; he is surrounded by several assistants and messengers. One of his duties is to maintain a uniform speed of progress in all parts of the globe. In this respect he is like the conductor of an orchestra in which the instruments are slide rules and calculating machines. But instead of waving a baton he turns a beam of rosy light upon any region that is running ahead of the rest, and a beam of blue light upon those who are behindhand."

evaluations, specific benchmarks have been developed for evaluating shared- and distributed-memory parallel computers. These vary from simple parallel loops, which measure the abilities of parallelizing compilers, to the PERFECT benchmark, which consists of 13 programs (including several fluid dynamics programs), and MIMD benchmarks such as Genesis, which consists of programs for fast Fourier transforms, molecular dynamics, linear algebra and numerical solutions of elliptic partial differential equations.

Measures of performance based on Amdahl's law are particularly effective for small programs that do not require extensive and intensive use of computer memory. Most programs used as computer benchmarks are of this sort, but they do not represent many of the requirements for the solution of grand challenge problems like turbulence simulation. For example, we can now simulate a field of homogeneous turbulence at Reynolds numbers comparable to those of low-turbulence-level laboratory wind tunnels in one day on a 50-megaflop, 32-megaword desk-side superworkstation using 128^3 modes. In 1970 such a computation would have required many months on the CDC 7600 supercomputer even though the peak CPU speed of the CDC 7600 was also roughly 50 megaflops. This marked difference in throughput is due mainly to the limited memory size of the CDC 7600, which would have made necessary many slow data transfers to disk.

We believe the issues of balancing memory, network speed and processing speed in computer design are best addressed by examining the Prototype Parallel Computer, depicted in figure 3, which we designed to solve a three-dimensional fluid dynamics problem. The key components of the PPC are an interconnecting set of P processing elements with distributed local memories, a shared global memory and a fast disk system. To avoid computational bottlenecks, data must be transferable among these components in roughly comparable times. We start by considering memory size, because we envision that grand challenge problems will have the computer fully dedicated to them for periods of 10^6 seconds or so (roughly two weeks) per run. This situation is quite different from that of running a shared resource at a computer center, in which many jobs contend for resources simultaneously.

Let us assume that N^3 modes are used to resolve the flow field ($N = 1024$, for example, will be possible within the next two years). The total memory required (including all three velocity components, pressure and various history data) is then $K_D N^3$ for some constant K_D of order 10, so we require the disk system to have memory size $M_D \geqslant K_D N^3$. The shared memory is assumed to be large enough to hold several dozen two-dimensional planes of data, so that its size $M_S \geqslant K_S N^2$, where K_S is at least 3–10 times the number of planes of data stored in the shared memory. Finally, the local memories must be large enough to hold several "pencils" of one-dimensional data, so their size $M_L \geqslant K_L N$, where K_L is 3–10 times the number of pencils stored in each local memory. (The values of these K factors depend on the number of variables needed at each mode for the most memory-intensive steps of the computation and on the latency time of the storage device at the next higher level.) If we assume that the size of the shared memory is P times that of the local memories, that is, $M_S \approx PM_L$, then we can avoid discussions of the detailed architectural interconnections among processors of the PPC.

Next we assume that a total of γN^3 computations are required per time step, where γ is the number of operations per mode (or grid point) per time step. In fluid dynamics computations γ is usually of order 250–5000, depending on the algorithm. Here γ is a measure of the computational complexity of the numerical method used to solve the flow equations (see the discussion in the next section). We assume that the code is highly parallelizable and does not suffer from inefficiencies due to parallelization; that is, we assume $E_P \approx 1$.

To proceed with the design of the PPC for our turbulence problem we first choose M_D and M_S as described above. Next we choose the number of processors P so that the computations can be accomplished in 10^6 seconds. That is, we choose P so that $N_t \gamma N^3 \leqslant 10^6 PS$, where N_t is the number of time steps required and S is the speed of each processor in flops. Typically $N_t \approx 100N$. For example, in the immediate future we can envisage $S = 100$ megaflops and $N = 1024$, so that more than 1000 processors will be required.

Each time step of the computation takes $\gamma N^3/PS$ seconds, and in an efficient design all data transfers must also be completed in that time. If data are transferred between each processor and local memory at speed μ_{PL} words per second, between each local memory and shared memory at a speed μ_{LS}, and between shared memory and fast disk at speed μ_{SD}; and if at each time step there are $Q_{SD} N^3$ words transferred between disk and shared memory, and a total of $Q_{LS} N^3$ words transferred between all local memories and shared memory, then we require

$$Q_{SD} \frac{N^3}{\mu_{SD}} \approx Q_{LS} \frac{N^3}{P\mu_{LS}} \approx \gamma \frac{N^3}{P\sigma\mu_{PL}} \approx \gamma \frac{N^3}{PS} \qquad (4)$$

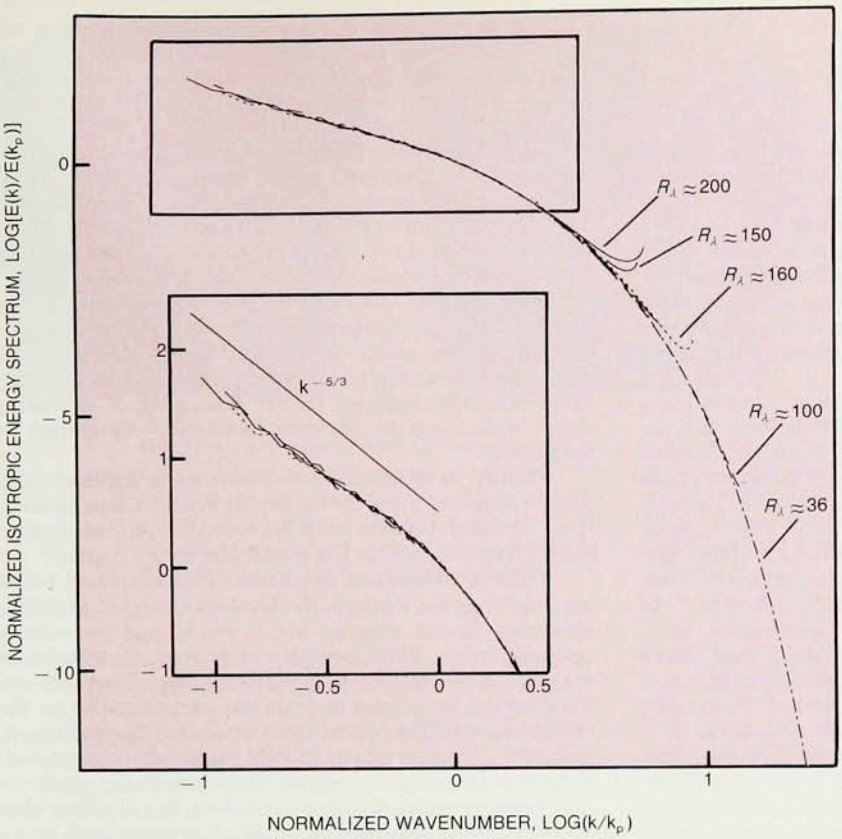
where $\sigma \approx 1$–2 is the typical number of operations that a processing element performs on each word of data that is transferred to it from a local memory. Thus, with $S = 100$ megaflops, $P = 1000$, $\gamma = 1000$, $Q_{SD} = 20$ and $Q_{LS} = 50$ (typical values for a spectral turbulence simulation), we must have $\mu_{SD} \approx 15$ gigabytes/sec, $\mu_{LS} \approx 40$ megabytes/sec and $\mu_{PL} \approx 800$ megabytes/sec. If $K_D = 10$ and $N = 1024$, then the disk size must be at least $M_D = 100$ gigabytes, while M_S and PM_L may be an order of magnitude or more smaller.

The principal conclusion from this analysis using the PPC model is that the solution of these large DNS problems requires a correspondingly large storage device (a fast disk in the case of the PPC) with a high transfer rate between the corresponding storage components. One must scale up the numbers given in this example to estimate performance requirements for an efficient and effective teraflop multiprocessor computer.

Modes: Discrete approximations to flows

Just as supercomputer architectures have undergone significant changes roughly every 20 years, so too have the numerical methods that solve incompressible- and compressible-flow problems. Early work was based almost exclusively on finite-difference methods, which approximate derivatives by discrete differences. Then in the 1960s, finite-element methods (based on variational formulations in terms of piecewise polynomial representations of the solution) came to the fore. Spectral methods, discussed below, underwent significant development through the 1970s and '80s, and most current work on the direct numerical simulation of turbulence uses them. Today the emphasis is on combining the best features of all the previous methods to yield efficient and accurate hybrid flow solvers.

We distinguish between methods that have been used primarily for simulations of incompressible turbulence and methods that have been used for simulations of compressible turbulence containing shock waves, which typically require special treatment. For incompressible flows we discuss spectral, spectral-element and particle



Andrei Kolmogorov's scaling law is verified by direct numerical simulations of turbulence using a 512^3 spectral code on the CM-200 at Los Alamos. Isotropic energy spectra at various Taylor microscale Reynolds numbers R_λ were rescaled by the maximum dissipation wavenumber k_p and $E(k_p)$ to give the plot shown here. The $R_\lambda \approx 150$ line was obtained using a different (time-independent) forcing term. The inset at the lower left expands the vertical scale of the area in the rectangle to show the close agreement of the data with the slope ($k^{-5/3}$) predicted by Kolmogorov's law. (Courtesy of Zhen-Su She, University of Arizona; Shiyi Chen and Gary D. Doolen, Los Alamos National Laboratory; and Robert H. Kraichnan.) **Figure 4**

methods, while for compressible flows we discuss hybrid finite-difference methods, including flux-corrected transport and piecewise parabolic methods. All these methods have been used for direct numerical simulation and large-eddy simulation of turbulence.

Spectral methods. In spectral methods the Navier–Stokes equations are solved using series expansions in terms of smooth functions such as complex exponentials and orthogonal polynomials. The first direct numerical simulation of homogeneous, isotropic turbulence[3] used a Fourier series representation of solutions of the incompressible Navier–Stokes equation in a periodic box with 32^3 modes. Fast transform techniques were employed to move freely between Fourier and physical space representations of fields. The computational complexity of this spectral algorithm is relatively low; for $N \lesssim 1000$ we obtain $\gamma \approx 500$, and more than 80% of the CPU time is spent on fast Fourier transforms. The key computational kernels (or code segments) are the fast Fourier transforms and the array transposes necessary to access different spatial directions.

Let us illustrate these points by outlining how such a spectral computer code is designed to solve the time-discretized Navier–Stokes equations,

$$\frac{\mathbf{v}^{n+1}-\mathbf{v}^{n-1}}{2\Delta t} = \mathbf{v}^n \times \boldsymbol{\omega}^n - \nabla \Pi + \frac{\nu(\nabla^2 \mathbf{v}^{n+1} + \nabla^2 \mathbf{v}^{n-1})}{2} \quad (5)$$

$$\nabla \cdot \mathbf{v}^{n+1} = 0 \quad (6)$$

where Δt is the time step, $\mathbf{v}^n$ is the velocity field at time step n, and $\boldsymbol{\omega}^n = \nabla \times \mathbf{v}^n$ is the vorticity field. The time-stepping scheme used in equation 5 leads to errors of order $(\Delta t)^2$. At the start of a time step we assume that $\mathbf{v}^n$ and $\mathbf{v}^{n-1}$ are stored on the disk of the PPC in terms of their complex Fourier coefficients $\mathbf{v}^n(k,p,q)$ and $\mathbf{v}^{n-1}(k,p,q)$. The momenta (k,p,q) are the (x,y,z) wavenumbers. The stages of the computation are given in the box on page 40. (See also figure 3.)

By optimizing memory allocations in the algorithm shown in the box it is possible to achieve a parallel implementation with $K \approx 6$ and $Q_{SD} = 18$. Such a spectral code with $N = 512$ currently runs at 20 seconds per time step in 32-bit precision on a 512-processor Intel Delta computer[14] (30 times faster than on a single-processor Cray YMP) and at 30 seconds per time step on a 64-kilobyte CM-200.[15] These speeds, however, are less than one-third of the code's theoretical peak speeds on these computers because of interprocessor communication and memory access delays, so that these machine resources are not quite balanced according to the criteria developed for the PPC.

Similar spectral codes are now routinely used[4] to study boundary-layer flows and flows in channels using Fourier representations parallel to the boundary but using Chebyshev or Jacobi polynomials in the inhomogeneous directions. For these problems the required computational kernels include fast Fourier transforms, direct matrix–vector multiplications, and inversions of tridiagonal matrices (matrices whose only nonzero elements are on the diagonal and adjacent to it). The corresponding complexity measure is $\gamma \approx 800$.

In the past decade spectral methods have been extended to problems in complex geometries, such as flow past a sphere. (See the cover of this issue.)

Spectral-element methods[16] combine some of the best features of spectral methods with those of finite-element methods by decomposing the domain into subdomains within which the variables and geometry are

represented as high-order tensor products of spectral polynomials. In this approach there is only a weak coupling between the dependent variables of adjacent subdomains, resulting in relatively sparse matrices that must be solved. The latter feature is critical to keeping the memory requirements and the processing time, and hence the computational complexity, of the method within reasonable bounds. In addition, the intrinsic coarse granularity (the "domain decomposition") of spectral-element methods leads naturally to a geometry-based distribution of work among processors that allows a high degree of parallelism.[17] The key computational kernels are scalar products, matrix–vector multiplications and matrix–matrix multiplications. The corresponding value for γ is approximately 2500.

Particle methods have been used for simulating a variety of incompressible and compressible flows and for plasma simulations (see the article by John M. Dawson, Victor Decyk, Richard Sydora and Paulett Liewer on page 64). For incompressible flows two types of particle methods are popular: vortex methods and lattice gases. Random vortex methods have been used to simulate high-Reynolds-number, mostly incompressible, turbulent flows, including shear flows of chemically reacting species.[18] In methods of this sort vorticity is approximated by a collection of particles (or "vortex blobs") that carry discrete quantities of vorticity. The corresponding velocity field is obtained from the vorticity field by the Biot–Savart law (by analogy with the deduction of a magnetic field from underlying current loops). The computational kernels involve the solution of N-body problems for the interior of the domain and on the boundary of the flow, and the solution of a potential-flow problem to guarantee that the induced vorticity does not cause flow across the boundary. In addition, viscous effects require the dynamic generation of vortex elements at the boundary to impose the condition that fluid does not slip along the wall at the boundary.

Lattice methods, including lattice gases and lattice versions of the Boltzmann and Bhatnagar–Gross–Krook (BGK) kinetic equations,[19] are intrinsically parallelizable due to local interactions and communications. They involve a novel statistical mechanics of discrete particles with discrete velocities whose average coarse-grained behavior follows the Navier–Stokes equations. These methods are particularly effective in treating highly complex flows, such as porous media flows, multiphase flows and flows over rough boundaries. Recently there has been interest in the possibility of extending these techniques to perform large-eddy simulation of turbulence.

Finally, it is possible to combine the application of these lattice or other low-order finite-difference descriptions in local regions with high-order spectral-element descriptions applied in the remainder of the region.[16]

Hybrid difference methods. Flux-corrected transport methods were originally developed to treat problems involving strong shocks, blast waves and chemically reactive flows. More recently they have been used in simulating compressible turbulent flows. They enforce the physical principles of positivity and causality on the numerical solution of problems involving sharp discontinuities.[20] These methods modify relatively conventional difference methods for incorporating hyperbolic conservation laws by using solution-dependent flux limiters that prevent the appearance of artificial extrema and hence artificial oscillations in the solution. Three-dimensional compressible codes are developed using one-dimensional subroutines; this is justified mathematically by factoring evolution operators ("directional splitting"). A three-dimensional computation requires roughly 30 calls to one-dimensional subroutines. The computational complexity is $\gamma \approx 2500$.

The piecewise parabolic method[21] is a hybrid scheme that combines classical difference methods and high-order interpolation techniques constrained so that sharp flow features are resolved using only about two computational cells. In this method there is no explicit incorporation of viscous dissipation; instead dissipation is introduced at high wavenumbers by discretization errors that arise in approximating the inviscid Euler equations.[21] The scheme also uses directional splitting. Subdomains, typically three-dimensional bricks that constitute a part of a three-dimensional uniform grid, are assigned to individual nodes. The computational and data-communication complexity of the piecewise parabolic method is due to local finite-difference arithmetic and transfer of the five primitive variables residing along edges of the subdomains. The computational complexity is $\gamma \approx 2500$.

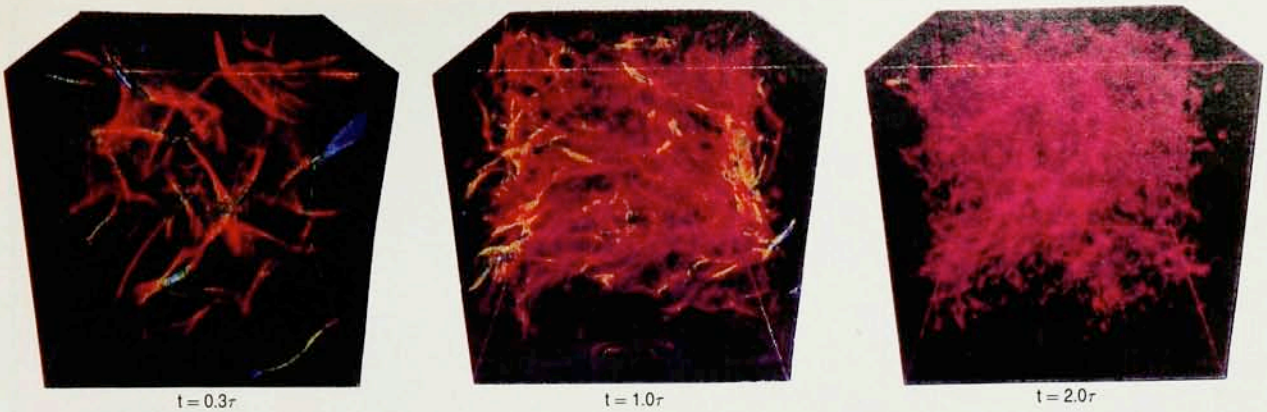
Steps in Typical Parallel Spectral Program

1. Import x–y planes of $\mathbf{v}^n$ from disk storage (DS) to shared memory (SM).
2. Import x-pencils from SM to local memory (LM) and compute x-fast Fourier transform (FFT) of $\mathbf{v}^n$ and $\boldsymbol{\omega}^n$. Result: $\mathbf{v}^n(x,p,q)$, $\boldsymbol{\omega}^n(x,p,q)$.
3. Export results of step 2 from LM to SM.
4. Import y-pencils from SM to LM and compute y-FFT. Result: $\mathbf{v}^n(x,y,q)$, $\boldsymbol{\omega}^n(x,y,q)$.
5. Export results of step 4 from LM to SM to DS.
6. Import x–z planes from DS to SM.
7. Import z-pencils from SM to LM and compute z-FFT. Result: $\mathbf{v}^n(x,y,z)$, $\boldsymbol{\omega}^n(x,y,z)$. Then compute $\mathbf{r} = \mathbf{v}^n \times \boldsymbol{\omega}^n$ in physical space and perform inverse z-FFT of $\mathbf{r}$. Result: $\mathbf{r}(x,y,q)$.
8. Export $\mathbf{r}$ from LM to SM to DS.
9. Import x–y planes of $\mathbf{r}(x,y,q)$ and $\mathbf{v}^{n-1}(x,y,z)$ from DS to SM.
10. Import x-pencils of $\mathbf{r}(x,y,q)$ from SM to LM and compute inverse x-FFT. Result: $\mathbf{r}(k,y,q)$.
11. Export results of step 10 from LM to SM.
12. Import y-pencils of $\mathbf{r}(k,y,q)$ from SM to LM and compute inverse y-FFT of $\mathbf{r}$. Result: $\mathbf{r}(k,p,q)$.
13. Solve for Π algebraically to impose incompressibility: $\Pi(k,p,q) = -i(kr_1 + pr_2 + qr_3)/(k^2 + p^2 + q^2)$. (This equation is derived by applying equation 6 to equation 5 and Fourier-transforming.)
14. Import $\mathbf{v}^{n-1}(k,p,q)$ from SM to LM and evaluate $\mathbf{v}^{n+1}(k,p,q)$ using the Fourier transform of equation 5. Result: $\mathbf{v}^{n+1}(k,p,q)$.
15. Export $\mathbf{v}^{n+1}$ from LM to SM to DS, completing the time-step cycle.

Flow codes: Parallel simulations of turbulence

We now briefly describe four applications of parallel computers to turbulent flow problems; the first two involve incompressible flows, while the latter two involve compressible supersonic flows.

Homogeneous turbulence. A 512^3 spectral simulation[15] has been performed on the 64K CM-200 SIMD parallel computer to verify Kolmogorov's theory of small eddies. (See the article by Frisch and Orszag.) With this high resolution it was possible to simulate homogeneous turbulence with confidence up to Taylor microscale Reynolds numbers $R_\lambda \approx 200$. In figure 4 we give a log–log plot of the energy spectra, rescaled by a characteristic



Decaying supersonic turbulence simulated using a three-dimensional piecewise parabolic method on the CM-5. Colors indicate normalized pressure, with values increasing from red to yellow to green to blue; τ is the time it takes a sound wave to propagate across the periodic computational box. The volume rendering is based on an opacity proportional to the negative velocity divergence, so that regions near shock waves are most opaque. (Courtesy of David H. Porter and Paul R. Woodward, University of Minnesota; and Annick Pouquet, Observatoire de la Côte d'Azur.) **Figure 5**

dissipation wavenumber k_p, for several Reynolds numbers R_λ. The results plotted in this figure show that Kolmogorov's universal scaling theory collapses all the data to a single curve and thereby gives an accurate description of turbulence energetics.

Drag reduction by riblets. One of the more interesting methods for reducing boundary-layer drag uses "riblets"—microgrooves aligned with the mean flow direction. The skins of some species of fast-swimming sharks have riblets. Riblets were successfully employed in the 1987 America's Cup competition and have already been tested at flight conditions. It has been found that riblets can reduce drag by 4–12% for flow over a flat plate. However, no clear explanation of the mechanism of turbulent drag reduction by riblets has yet been confirmed. To advance the understanding and expedite the design, placement and shape of riblets, direct numerical simulation of flows with riblets have been performed using a hybrid spectral-element–Fourier spectral method on the Intel Gamma and Delta Touchstone parallel computers.[22] With 512 processors, speeds in excess of 3 gigaflops are obtained (3 seconds per time step for 100 elements of resolution $10\times10\times256$). Figure 1 shows the instantaneous streamwise velocity component of the three-dimensional flow field at three different cross-flow planes. The simultaneous visualization of flow structures on the upper (smooth) wall and the lower (riblet) wall leads to quantitative predictions and to a qualitative model of the turbulence production and associated shear stress.

Supersonic, compressible homogeneous turbulence. High-resolution (up to 512^3) simulations of supersonic homogeneous turbulence have been carried out on the parallel CM-5 computer using the piecewise parabolic method[21] and on the Intel Touchstone prototype using a sixth-order finite-difference method.[23] For the CM-5 code the data are partitioned into 512 blocks mapped onto 512 nodes; the code runs at approximately 1.5 gigaflops using only the scalar CM-5 chips. Figure 5 shows a perspective volume rendering of the pressure field of a turbulence decay run. The simulation begins with a field of homogeneous turbulence with rms Mach number 1.1; the goal is to see how shock waves develop as the turbulence dissipates. The figure shows the pressure at times 0.3τ, 1.0τ and 2.0τ, where τ is the time that it takes a sound wave to propagate across the computational box. Apparently the number of shocks increases and the typical shock strength decreases with time, although there are still some fairly large pressure jumps even at later times. Such simulations show that in a supersonic flow vorticity is produced by shock curvature and shock intersections rather than by the random vortex stretching mechanism that is dominant in subsonic and incompressible flows.

Supersonic reacting shear layer. Parallel flux-corrected transport computations of supersonic, multispecies, chemically reacting, exothermic turbulent flows have run at 800 megaflops on the CM-200 with 16K processors and have been used to evaluate new concepts for high-speed propulsion.[24] Figure 6 shows the hydrogen mole fraction at an advanced stage in the mixing of two counterflowing supersonic streams of hydrogen and air in a small ($1\,\text{cm}\times1\,\text{cm}$) region. Such conditions might be found in the engine of the proposed National Aerospace Plane. Because the computations involve nine species undergoing physicochemical processes (including convection, thermal conduction and chemical reactions), they tax the capabilities of the most powerful parallel computers.

Perspective

Experience has shown that each time a new supercomputer is introduced, it takes several years for software to mature on the new architecture, and usually by the time the software has matured, new versions of the computer system are available. Nevertheless, it has been possible to make effective use of the new architectures at an early date for computational fluid dynamics (CFD), even without effective, general purpose software. In fact, it is in the early years of new architectures that many of the most important scientific discoveries occur. To achieve such results, one must understand the basic computer architecture and its optimal use, which may require using low-level (even assembly) languages. The knowledge gained in these leading-edge CFD applications has been of direct benefit to developers of compilers and higher-level languages. Effective collaborations between CFD scientists and computer hardware and software experts will be critical to the development of the new teraflop computer environments.

Electronic component speeds and densities have improved by a factor of more than 10^5 in the last half-century. This development is unrivaled in other fields of

Supersonic streams of counterflowing, chemically reacting hydrogen and air simulated by flux-corrected transport on the CM-2. Color indicates hydrogen mole fraction: Deep red represents pure hydrogen; deep purple, pure air. The lower flow is moving from left to right. (Courtesy of Patrick Vuillermoz, ONERA; and Elaine Oran, Naval Research Laboratory.)
Figure 6

human endeavor; if automobiles had undergone similar improvements, today a Cadillac would sell for less than a penny, or it would be capable of a peak speed in excess of 1% of the speed of light, or one gallon of gas would suffice for about ten trips to the Moon. Despite these remarkable advances in computer electronics, the motivating force behind computer developments has been (and will likely continue to be) the grand challenge applications. Indeed, it was the application of numerical weather forecasting that inspired the British meteorologist Lewis Fry Richardson in 1922 to foresee the use of MIMD parallel computers. (See the box on page 37.)

In the same way, the foresight of CFD scientists following in Richardson's tradition will likely drive many of the most significant future computer developments. We expect that continued development of hybrid numerical methods, in conjunction with the development of physical models (based on fundamental theory and integrated with the results of prototype experiments) and the consideration of computer architectures like the Prototype Parallel Computer, will form the basis for breakthroughs on the grand challenges in fluid mechanics.

* * *

We would like to acknowledge our colleagues, too numerous to mention here, who have provided us with up-to-date information in this rapidly developing field.

References

1. "Grand Challenges 1993: High Performance Computing and Communications," report by the Committee on Physical, Mathematical and Engineering Sciences, Federal Coordinating Council for Science, Engineering and Technology, Washington, D. C. (1992).
2. H. W. Emmons, Annu. Rev. Fluid Mech. **2**, 15 (1970).
3. S. A. Orszag, G. S. Patterson, Phys. Rev. Lett. **28**, 76 (1972).
4. M. Y. Hussaini, R. G. Voigt, eds., *Instability and Transition*, vols. I and II, Springer-Verlag, New York (1990). J. Kim, P. Moin, R. Moser, J. Fluid Mech. **177**, 133 (1987). P. Spalart, J. Fluid Mech. **187**, 61 (1988).
5. G. E. Karniadakis, Appl. Num. Math. **6**, 85 (1989). L. Kaiktsis, G. E. Karniadakis, S. A. Orszag, J. Fluid Mech. **231**, 501 (1991). A. G. Tomboulides, S. A. Orszag, G. E. Karniadakis, preprint AIAA-93-0546, Am. Inst. of Aeronautics and Astronautics, New York (January 1993).
6. B. Galperin, S. A. Orszag, eds., *Large Eddy Simulations of Complex Engineering and Geophysical Flows*, Cambridge U. P., New York (1993).
7. S. A. Orszag, V. Yakhot, W. S. Flannery, F. Boysan, D. Choudhury, J. Maruzewski, B. Patel, in *Near-Wall Turbulent Flows*, R. M. So, C. G. Speziale, B. E. Launder, eds., Elsevier, New York (1993).
8. A. Jameson, Science **245**, 361 (1989); Aerospace America **30**, 42 (1992). R. K. Agarwal, J. C. Lewis, in *Symp. on High Performance Computing for Flight Vehicles*, Washington, D. C., 7–9 December 1992, in press. M. Y. Hussaini, in *11th Int. Conf. on Numerical Methods in Fluid Dynamics*, D. L. Dwoyer, M. Y. Hussaini, R. G. Voigt, eds., Springer-Verlag, New York (1989), p. 3.
9. A. J. Smits, Exp. Thermal Fluid Sci. **5**, 579 (1992).
10. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan and Kaufmann, San Mateo, Calif. (1992). See also IEEE Spectrum, September 1992.
11. S. L. Johnsson, in *Topics in Atmospheric and Oceanic Sciences*, Springer-Verlag, New York (1990), p. 231.
12. G. E. Amdahl, in *Proc. AFIPS Spring Joint Computer Conf.*, Atlantic City, N. J., 18–20 April 1967, Thompson, Washington, D. C. (1967), p. 483.
13. J. Dongarra, W. Gentzsch, Parallel Computing **17**, 1067 (1991).
14. A. Wray, R. Rogallo, "Simulation of Turbulence on the Intel Delta Gamma," NASA Technical Memorandum, April 1992.
15. S. Chen, G. D. Doolen, R. H. Kraichnan, Z.-S. She, Phys. Fluids A **5**, 458 (1993). Z.-S. She, S. Chen, G. D. Doolen, R. H. Kraichnan, S. A. Orszag, submitted to Phys. Rev. Lett. (1993).
16. G. E. Karniadakis, S. A. Orszag, E. M. Ronquist, A. T. Patera, in *Incompressible Fluid Dynamics*, M. D. Gunzburger, R. A. Nicolaides, eds., Cambridge U. P., New York (1993), in press. G. E. Karniadakis, S. A. Orszag, in *Algorithmic Trends for Computational Fluid Dynamics*, M. Y. Hussaini, A. Kumar, M. Salas, eds., Springer-Verlag, New York (1993), in press.
17. P. Fischer, A. T. Patera, J. Comput. Phys. **92**, 380 (1991).
18. A. J. Chorin, J. Comput. Phys. **27**, 428 (1978). A. Leonard, J. Comput. Phys. **37**, 289 (1980). J. A. Sethian, A. F. Ghoniem, J. Comput. Phys. **74**, 283 (1988).
19. G. Doolen, ed., *Lattice Gas Methods for Partial Differential Equations*, Addison–Wesley, Redwood City, Calif. (1989). F. Higuera, S. Succi, R. Benzi, Europhys. Lett. **9**, 663 (1989). Y. H. Qian, D. d'Humieres, P. Lallemand, Europhys. Lett. **17**, 479 (1992).
20. E. S. Oran, J. P. Boris, *Numerical Simulation of Reactive Flow*, Elsevier, New York (1987).
21. D. H. Porter, A. Pouquet, P. R. Woodward, Theor. Comput. Fluid Dynamics **4**, 13 (1992).
22. D. Chu, R. D. Henderson, G. E. Karniadakis, Theor. Comput. Fluid Dynamics **3**, 219 (1992). R. D. Henderson, PhD thesis, Princeton U., Princeton, N. J. (1993).
23. G. Erlebacher, private communication (1992).
24. E. S. Oran, J. P. Boris, C. R. Devore, J. Fluid Dynamics Res. **10**, 251 (1992). ■