

Very High Resolution Simulation of Compressible Turbulence on the IBM-SP System*

A. A. Mirin[†] R. H. Cohen[†] B. C. Curtis[†] W. P. Dannevik[†]
A. M. Dimits[†] M. A. Duchaineau[†] D. E. Eliason[†] D. R. Schikore[†]
S. E. Anderson[‡] D. H. Porter[‡] P. R. Woodward[‡] L. J. Shieh[§]
S. W. White[§]

Abstract

Understanding turbulence and mix in compressible flows is of fundamental importance to real-world applications such as chemical combustion and supernova evolution. The ability to run in three dimensions and at very high resolution is required for the simulation to accurately represent the interaction of the various length scales, and consequently, the reactivity of the intermixing species. Toward this end, we have carried out a very high resolution (over 8 billion zones) 3-D simulation of the Richtmyer-Meshkov instability and turbulent mixing on the IBM Sustained Stewardship TeraOp (SST) system, developed under the auspices of the Department of Energy (DOE) Accelerated Strategic Computing Initiative (ASCI) and located at Lawrence Livermore National Laboratory. We have also undertaken an even higher resolution proof-of-principle calculation (over 24 billion zones) on 5832 processors of the IBM system, which executed for over an hour at a sustained rate of 1.05 Tflop/s, as well as a short calculation with a modified algorithm that achieved a sustained rate of 1.18 Tflop/s. The full production scientific simulation, using a further modified algorithm, ran for 27,000 timesteps in slightly over a week of wall time using 3840 processors of the IBM system, clocking a sustained throughput of roughly 0.6 teraflop per second (32-bit arithmetic). Nearly 300,000 graphics files comprising over three terabytes of data were produced and post-processed. The capability of running in 3-D at high resolution enabled us to get a more accurate and detailed picture of the fluid-flow structure - in particular, to simulate the development of fine scale structures from the interactions of long- and short-wavelength phenomena, to elucidate differences between two-dimensional and three-dimensional turbulence, to explore a conjecture regarding the transition from unstable flow to fully developed turbulence with increasing Reynolds number, and to ascertain convergence of the computed solution with respect to mesh resolution.

1 Introduction

The simulation of turbulence and mix has been one of the most demanding challenges of computational hydrodynamics. Many real-world applications (e.g., chemical combustion, supernova evolution, supersonic transport) are strongly dependent on the extent and structure of interpenetration of disparate fluids. For example, the characteristic scales of the interpenetrating species affect the surface area to volume ratio, and hence the reactivity of the species. Often there is such a wide range of length scales that very high resolution is required for the simulation to accurately reflect how these scales interact. Moreover, for turbulent flows that are inherently three-dimensional, the two-dimensional approximation is often inadequate because of a tendency for those flows to develop an inappropriate inverse energy cascade. Highly compressible flows provide an even greater

*This is LLNL Report UCRL-JC-134237.

[†]Lawrence Livermore National Laboratory, Livermore, CA

[‡]University of Minnesota, Minneapolis, MN

[§]IBM, Austin, TX

challenge, as the algorithms are required to both preserve shocks yet not overly dissipate the smooth flow.

The work presented here deals with the numerical simulation of compressible three-dimensional flows with shocks, using spatial resolutions never before considered feasible. We utilize the simplified Piecewise Parabolic Method (sPPM) code of Woodward, et al. [1] on the IBM-SP Sustained Stewardship TeraOp (SST) machine at Lawrence Livermore National Laboratory (LLNL). We first present a proof-of-principle calculation that executed at a sustained 1.05 Tflop/s rate for over an hour. We then present a shorter proof-of-principle test that, using a modified algorithmic kernel, executed at 1.18 Tflop/s, and then a third test that used over 70 billion computational zones.

The focus of this effort, though, is on a scientific simulation of the Richtmyer-Meshkov instability [2], which comes about when a shock intersects a contact discontinuity. The simulation presented here, which made use of a further modified algorithmic kernel, executed for 27,000 timesteps using over 8 billion computational zones at a sustained throughput of 0.6 Tflop/s (32-bit arithmetic). The project involved not only executing the code, but producing, offloading and postprocessing nearly 300,000 files containing graphics data, whose total storage exceeded 3 terabytes.

Being able to run at such high resolution enabled us to observe relevant effects that could not be addressed at lower resolution. This allowed us to provide support for an important conjecture involving the development of turbulence, and to demonstrate how the interaction of long and short length scales can lead to the development of an even finer scale structure.

Section 2 discusses the sPPM code. Section 3 describes the IBM platform used for this study. In section 4 we present the proof-of-principle tests. Section 5 describes the scientific problem, its execution on the IBM system, and file retrieval. Section 6 addresses postprocessing and interpretation of the data. Conclusions are presented in section 7.

2 The sPPM code

2.1 Computational algorithm

The sPPM code solves the compressible Euler equations using a simplified implementation of the Piecewise Parabolic Method (PPM), which is a high-order accurate Godunov method developed by Colella and Woodward [3]. In a study that helped drive the development of the PPM scheme, Woodward and Colella [4] compared PPM to several other difference methods for problems involving strong shocks. Use of the Godunov approach in PPM makes this numerical scheme upstream-centered in each Riemann invariant separately. Together with nonlinear solutions of Riemann's shock tube problem at grid cell interfaces when strong waves are present, this upstream centering produces sharp numerical representations of shocks on the computational grid. Monotonicity constraints inspired by the work of van Leer with the MUSCL scheme [5], together with a numerical diffusion term that adapts to the conditions of the local flow, keep the sharp shock fronts of the PPM scheme from generating unwanted and unphysical noise in the solution. PPM also includes an interpolation scheme that is fourth order accurate for small timesteps. This interpolation scheme detects contact discontinuities in the flow solution and, when they are present, it employs an alternative interpolation technique that helps to keep the numerical representation of these discontinuities sharp. A library of PPM code modules, suitable for use in parallel computation, is being made available by the Laboratory for Computational Science and Engineering at the University of Minnesota under support from the DOE Office of Science [6].

SPPM contains several features of PPM, but significant ones have been omitted. Among those omitted features are contact discontinuity detection and steepening, and the computation of a coefficient of numerical viscosity that adjusts to the local needs of the flow in both space and time. This particular implementation of sPPM makes use of a Lagrangian method combined with a remap onto the original mesh, so that the algorithm is effectively Eulerian. The fluid is assumed to obey a single gamma-law equation of state. The time integration makes use of directional splitting along the three coordinate axes, with the order of directions reversing each timestep.

2.2 Parallel implementation

The sPPM code is written in Fortran 77 with some C routines. It uses a logically rectangular three-dimensional domain decomposition. Each subdomain is mapped to a computational node of the parallel architecture on which the code is being executed. Message-passing between nodes is accomplished using standard Message Passing Interface (MPI) calls. We have modified the sPPM driver to improve performance. The functionality has not changed (other than new restrictions on grid size), merely the programming implementation.

For each directional update, the three-dimensional mesh is mapped into pencils of $16 \times 16 \times N$ cells, with the long length, N , aligned along the sweep direction and the transverse directions tiled into 16×16 chunks. The logical pencils are processed in an order that facilitates overlap of communication with computation. A section of interior pencils (e.g., pencils not on the subdomain boundary) is processed first; this is concurrent with communication from the previous directional sweep. Next, the outer pencils (e.g., those closest to the subdomain boundary) are processed. The remaining interior pencils are then processed while the outer pencils are sending updated data to their transverse neighbors. After all pencil updates are complete, the new boundary data is communicated longitudinally (i.e., in the sweep direction) as the following directional sweep commences. Each pencil is updated with 256 separate calls to a controlling routine that updates a single 1-D strip. The stencil requires up to 5 cells distant in the principal direction and two in the transverse direction. Data is stored into temporary arrays for improved cache performance.

Multithreading is accomplished through POSIX threads. Additional slave threads are created, which along with the parent process communicate through shared static memory. Synchronization in shared memory is brought about through barriers implemented with atomic fetch-and-add operations. Parallel loops are self-scheduled using a shared integer index. One thread performs all of the message-passing (as well as participating in the strip updates); it assembles and disassembles message buffers and calls the MPI routines. The remaining threads are dedicated to the strip updates.

A second copy of the major data structure has been added so that the full range of data both before and after the one-dimensional sweep can be stored. This replaces the circular buffering that was used in the original driver. The availability of the second array cuts data movement in half and facilitates overlap of communication, though at increased storage cost. Parallelization in shared memory is with respect to pencils; strip updates within a pencil are performed sequentially.

3 IBM machine configuration

The IBM SST ASCI [7] Blue-Pacific system, located at Lawrence Livermore National Laboratory, is comprised of three 488-node sectors, with a total CPU count of 5856. Each node contains 1.5 to 2.5 Gbytes of local memory and is powered by four 332-MHz PowerPC 604e processors. Allowing for up to 2 operations per processor per clock period, the system peak performance is 3.9 TeraOp/s. The processor to memory bandwidth is 2.1 Tbyte/s (aggregate), and the node to node bandwidth is 150 Mbyte/s (bidirectional). There are 62.5 Tbytes of RAID storage, with an I/O bandwidth to local disk of 10.5 Gbyte/s (aggregate). The three sectors are connected by six High Performance Gateway Node (HPGN) switches, with a total of 2.4 Gbyte/s bandwidth (bidirectional).

4 Proof-of-principle tests

The proof-of-principle tests make use of the test problem provided with the sPPM benchmark. That problem involves a shock passing through a gas with a density discontinuity. The interaction of the shock and the discontinuity leads to the Richtmyer-Meshkov instability. These tests were performed prior to the delivery of the machine to LLNL and utilized 5832 of the total 5856 processors.

The first proof-of-principle calculation uses the standard sPPM benchmark (without kernel modification) on a $2304 \times 2304 \times 4608$ grid, partitioned into a $9 \times 9 \times 18$ domain decomposition, each subdomain supporting a 256-cubed grid. Four threads were assigned to each subdomain. The calculation ran for over an hour, and using 32-bit arithmetic achieved a 1.05 Tflop/s throughput. The flops were counted according to the rules governing the machine procurement, which assign 4 flops to floating divide and sqrt, and 1 flop to all other floating point operations.

An execution trace tool was used to get accurate counts. A single node run with a $256 \times 256 \times 256$ local grid was traced for 60 timesteps (same local grid and number of timesteps as the full-system run above), resulting in counts for every instruction executed. Then the non-floating-point instructions were discarded, and the contract rules were applied. We measured a total of 2904 billion floating point operations per node, which when combined with 1458 nodes and a 4026 second execution time gives 1.05 (32-bit) Tflop/s.

We also measured parallel performance. Within a four-processor node, memory contention reduces performance by about 10 percent, varying slightly with problem size, resulting in a factor of 3.6 speedup over single processor execution. Message-passing across nodes is almost entirely nearest-neighbor, resulting in very good scaling. When the local grid is sufficiently large, the communication is completely overlapped with computation, except for one global reduction per timestep. For 64 nodes arranged in a $4 \times 4 \times 4$ decomposition and a $256 \times 256 \times 256$ local grid, there is a 60.8-fold speedup over single node operation (applied to that same local grid), and a 218.9-fold speedup as compared to running with a single processor, for an 85.5 percent parallel efficiency. Maintaining the same local grid and increasing the node count beyond 64 incurs very little additional loss of parallel efficiency due to the locality of the communication. Running with our largest configuration of 1458 nodes (5832 processors) is only 2 percent slower than with 64 nodes, and has a parallel efficiency of 83.8 percent.

Next we used a version of the kernel that was heavily optimized by our IBM co-authors. The major optimizations included elimination of unnecessary/redundant operations, changing rarely needed unconditional operations to be conditional, moving some operations into the main routine, minimizing the number of vector temporaries to reduce the cache footprint, recoding and/or merging loops to improve pipelining, repartitioning loops to balance resource requirements, and making explicit calls to the MASS (Mathematical Acceleration SubSystem) library for vector reciprocal, sqrt and reciprocal sqrt functions.

We ran the same $2304 \times 2304 \times 4608$ problem using the same $9 \times 9 \times 18$ domain decomposition for 2 timesteps and achieved a throughput of 1.18 Tflop/s. The object of the optimized kernel was not this modest increase in flop rate, rather it was to reduce the wall time for the scientific calculation. The optimized kernel has approximately half the number of floating point operations as the original kernel, and the wall time was reduced by a factor of 2. The optimized kernel reduced the computation portion so dramatically that overlapping the communication became problematic, especially with smaller local grids.

Finally, we attempted to investigate the largest possible problem size that we could run on the IBM system. This involved using the original driver, since it is more memory efficient than the optimized driver, but is slower and does not overlap communication with computation. We also needed to use the original kernel, since the optimized kernel is incompatible with the original driver. We found that a problem size of $1460 \times 1460 \times 33215$, or 70.8 billion zones, would fit in the memory of the system. Since we had to use the slow driver and kernel, the throughput was only 0.88 Tflop/s, but even so the faster versions would not be fast enough to make a calculation of this size practical.

5 Simulation of the Richtmyer-Meshkov instability

5.1 Problem description

The scientific simulation approximates the shock tube experiment of Vetter and Sturtevant [8], in which two gases are initially separated by a membrane pushed against a wire mesh. The actual experiment uses air and sulfur hexafluoride, which in our calculation we approximate as a single gas having a specific heat ratio of $\gamma=1.3$. The contact discontinuity is located at $z=0.59$, where the shock tube extends from $z=0$ to $z=0.94$ (see Fig. 1). The densities to the left and right of the contact discontinuity are initially 1.0 and 4.88, respectively; the pressure equals 0.77 on either side. The calculation is carried out in a reference frame in which the contact discontinuity is roughly stationary; hence the fluid velocity on either side of the contact discontinuity is initialized to -0.47. Boundary conditions are periodic in the transverse (x and y) directions and continuation in the streamwise (z) direction; the latter does a good job of absorbing moderate-strength shocks.

The location of the contact discontinuity is initially perturbed. The perturbation is chosen to be

a superposition of a long wavelength and short wavelength disturbance, where the former represents the distortion of the mesh as it is being pushed, and the latter corresponds to the mesh spacing. Besides approximating the setup of an interesting experiment, this simulation provides information on the nonlinear interaction of two disparate scales, a fundamental building block for the nonlinear evolution of turbulence.

A shock of Mach 1.5 is applied from the low- z end. The initial shock position is at $z=0.47$, and the velocity difference across the shock is 0.73. The density and pressure behind the shock are 1.93 and 1.86, respectively.

5.2 Problem execution

The shock tube simulation was run on a $2048 \times 2048 \times 1920$ grid using 960 nodes of the IBM-SP system arranged in an $8 \times 8 \times 15$ domain decomposition, with four threads per subdomain, making each node responsible for a $256 \times 256 \times 128$ piece of the data. We were restricted in the number of nodes (as compared to the proof-of-principle tests) since only two of the three 488-node sectors were in place at LLNL. The particular domain decomposition and problem shape were chosen to maximize computational resources while staying within the constraint (since relaxed) of the local mesh being at least 128 and a multiple of 16 in each of the three directions. The problem was run for 27,000 timesteps - just over 9 transverse sound crossing times.

The simulation executed in 173 hours of machine time, spread over 226 hours of wall time, which was a remarkable efficiency given that the machine had just been delivered to Lawrence Livermore National Laboratory and was not yet generally available. Each node created its own component restart dump which was output to local disk, and immediately copied onto an additional node as backup. This occurred roughly every three hours. Occasionally a node would fail, and data was copied from the backup node onto a new node and the problem restarted. Each restart dump comprised 188 Gbyte (aggregate), and twice that amount was required to accommodate nodal backup components.

The sustained throughput was estimated to be 0.6 Tflop/s or greater. This estimate was obtained using an execution trace tool (essentially a simulator) applied to a $256 \times 256 \times 128$ grid, the same size grid for which a single node was responsible in the scientific simulation. We needed to restrict the performance measurement to the local grid size since the flop counting tool is serial. The results of the first two timesteps indicated an initial throughput of the scientific simulation of 0.627 Tflop/s. We then observed, using the trace tool, that conditional code in the kernel causes the flop count per timestep to increase as the execution proceeds. This increase occurs during the early phase of the calculation and asymptotes to about 21 percent. The wall time per timestep increases as well, but to a lesser extent than the flop count (16 percent). Therefore, the flop rate increases slightly as the calculation proceeds. Folding in an additional (observed) 6 percent degradation due to offloading and backup of scientific data, we cite the conservative figure of 0.6 Tflop/s for the sustained throughput.

The floating point computation rate in the science run is clearly lower than that of the proof-of-principle test carried out with the optimized kernel, even allowing for the lower node count. The figure of 1.18 Tflop/s on 1458 nodes would scale to 0.777 Tflop/s for the node count of 960 that was used in the science run. There are three main reasons for the lower rate. First, the science run utilized an additional optimization in the kernel, namely combining the vectorized square root and reciprocal functions into a vectorized reciprocal square root. This reduced wall time by 6 percent but reduced the flop count by 15 percent, resulting in a 9.6 percent decrease in flop rate. Second, the local grid for the science run was half as large as that of the proof-of-principle run. The smaller local grid has a less favorable surface-to-volume ratio, yielding less overlap of communication and computation, resulting in an 8 percent decrement in flop rate (recent optimizations in the communication strategy have since cut that difference to 3 percent). The final difference between the science run and the proof-of-principle run is that many system daemons were turned off for the latter. We can quantify that effect by noting that when the proof-of-principle test was rerun at LLNL, we consistently observed roughly 3 percent lower performance than prior to machine delivery; once the system daemons (most notably the Global Parallel File System daemon) were turned off, the flop rate returned to its pre-delivery value. Applying these three degradations to the 0.777 Tflop/s figure

yields 0.627 Tflop/s, in exact agreement with our throughput estimate for the first two timesteps.

Each node created its own component graphics data files (described below), in total comprising over 3 Tbytes of data, spread over 275,000 files. This data was output to local disk. Once there, it was compressed and copied (using parallel remote copies) onto the Global Parallel File System. The copying was under the control of nodes not being utilized for the sPPM execution. This procedure in effect accomplished asynchronous output, as the I/O took place during code execution with minimal effect (roughly 6 percent) on throughput. The data was then offloaded onto a graphics postprocessing machine and ultimately shipped to archival storage.

6 Postprocessing and interpretation of data

The sPPM code creates two types of graphics data files. Brick-of-byte (BOB) files contain one byte of information per gridpoint. The byte value (0 to 255) represents the value of the physical quantity concerned relative to the global minimum and maximum, subject to a user-specified transformation. The simulation reported on here utilized BOB files of entropy subject to a linear transformation, so that for example a byte value of 128 represented data halfway between the minimum and maximum. The code produced 274 such BOB dumps, each of aggregate length 8.1 Gbyte partitioned among 960 files.

The other type of output file is the “compressed dump.” Here all of the dynamical variables (e.g., density, pressure, velocity, material fraction) (or the logarithm for positive definite quantities such as density) are subjected to a linear transformation, and the representation is stored in 16-bit integer format. Ten compressed dumps were produced, each of aggregate length 80.6 Gbyte partitioned among 960 files. BOB files, which contain only partial and highly compressed information, are typically used for volume renderings, whereas compressed dumps may be used for more general purposes.

6.1 Volume rendering

The BOB data was postprocessed using the University of Minnesota volume renderer HVR (Hierarchical Volume Renderer) on an SGI Infinite Reality engine, in which infinite reality pipes are connected via high-bandwidth optical fiber channels to a high-capacity RAIDed array of Ciprico disks.

HVR, still in its development phase, is an outgrowth of the earlier volume renderers “perpath” (a serial, in core, software volume renderer) and “Bob” (an interactive extension of perpath for which both software and SGI texture-hardware versions exist), which were also developed at the University of Minnesota [9]. HVR is a highly efficient volume renderer which runs on, and fully utilizes, the texture hardware on SGI Infinite-Reality-Engine computers and which has additional special enhancements that enable it to make volume renderings at unprecedented graphical and data resolutions. These enhancements include the use of texture hardware, parallelization across multiple CPU’s and Reality-Engine pipes, and efficient out-of-core operation in which subregions of the data volume are rendered and these sub-renderings combined. This out-of-core, and out of texture-memory, operation involves fast reads of subsets of the data from a time slice from the high-capacity disk array.

A small number of renderings have been made from the full resolution data (2048 x 2048 x 1920 bytes) at a graphical resolution of 3840 x 3072 pixels. These images have been shown in presentations at full resolution on a 3840 x 3072 pixel SGI “Power Wall” projection system at LLNL. A larger number of lower resolution images have been produced to animate time evolution of the Richtmyer-Meshkov layer evolution in the simulation.

Figure 2 shows a volume rendering of entropy at the conclusion of the calculation. The shock has moved from left to right and a very small portion has reflected off the right-hand wall. As noted earlier, we are working in a reference frame in which the contact discontinuity, after being encountered by the shock, is approximately stationary. Figure 3 shows the entropy for a simulation at 384 x 384 x 384 resolution. The fine scale structure at high resolution is clearly not present at lower resolution. A simulation at 1024 x 1024 x 1024 (see Fig. 4) indicates structure similar in scale to that at 2048 x 2048 x 1920, whereas one at 192 x 192 x 192 (see Fig. 5) indicates structure

similar to that of $384 \times 384 \times 384$. Hence, there appears to be a transition from unstable flow with relatively large-scale structures, to turbulence. This is in accord with the conjecture of Dimotakis [10], who argues that such a transition should occur when there is sufficient separation between the energy-containing and dissipative scales to permit a well-developed forward cascade through an inertial range. Had we been restricted to lower resolution, we would not have been able to verify this transition in character. Also, comparison with visualizations of two-dimensional simulations at the same resolutions (see Fig. 6) provides a dramatic illustration of the difference between three-dimensional and two-dimensional dynamics (forward vs. inverse cascade); the two-dimensional runs, rather than developing random-looking fine-scale structure at high resolution, tend to remain characterized by extended structures but with sharper boundaries. Clearly, if the two fluid species had been reactive, the reaction rates in these two simulations would differ substantially.

With HVR one can “fly through” the volume. Such an excursion reveals a centrally-located cylindrical zone that contains identifiable bubbles and spikes that have survived for the duration of the simulation. These remnants of the initial short-wavelength perturbation are situated about the middle of the spike that originates from the initial long-wavelength perturbation. Outside of this cylinder (throughout most of the volume), the bubbles and spikes have given way to finer-scale turbulence. A comparison simulation of a single period of the short-wavelength perturbation shows a surviving bubble and spike, indicating that the coupling of long and short scales has destroyed most of the short-wavelength bubbles and spikes.

6.2 Compressed dump postprocessing

The compressed dumps are designed to be postprocessed by a3d, another member of the PPM Data Analysis and Rendering Toolkit [9]. A3d, which has been recently generalized to work in parallel and for large data sets, may be used to calculate any combination of algebraic, differential or integral forms on the raw data. The fields so produced may be output in the form of histograms, power spectra, cuts through the data, or brick-of-byte (BOB) files. Data is organized in a push-down stack (last in, first out) and input uses inverse Polish notation. A3d is designed to accept nodal input (one file per subdomain) and can convert between Big Endian and Little Endian formats. Like the volume renderer HVR, the unprecedented size of this data set has motivated the modification of a3d to run in parallel on SMP machines, run out-of-core on tiled subdomains, efficiently use and reuse first and second level cache, and utilize fast direct I/O, with I/O and computation overlapped.

Figure 7 shows z-velocity energy spectra near the midplane for three different resolutions. At the highest resolution, we observe an inertial range within the wave number domain of roughly 30 to 300. Because of the unprecedented resolution and the convergence trend as resolution is increased, we have confidence that this inertial range is meaningful. According to Dimotakis’ conjecture, it is the existence of this well-developed inertial range that allows for the observed transition of the small-scale structures to the turbulence shown in Fig. 2. Plots of the mixing layer extent for these three resolutions as well as comparison to experiment (see Fig. 8) indicate both convergence and good agreement between simulation and experiment. We could not have ascertained convergence - including a clear separation of the energy containing range, inertial range, and (numerically-induced) dissipation range - without the 8 billion zone simulation.

7 Conclusions

We have succeeded in carrying out a scientific calculation that until now was not feasible. The simulation used almost an order of magnitude higher resolution than previous successes and executed at a sustained 0.6 Tflop/s (32-bit arithmetic) for over a week (exclusive of machine down time). We were able to observe effects that could not be seen or verified at lower resolution; in particular we provided support for an important conjecture regarding the transition from large-scale unstable flow to fully developed turbulence, as well as evidence that the interaction of long and short wavelength perturbations destroys the short wavelength structures in favor of finer-scale turbulence. We believe this to be a significant advancement in our ability to simulate and understand turbulence.

We have been simulating turbulent flows for many years, but until now it has not generally been possible to resolve within a single computation both the large-scale flow and the turbulent

motions that it sets up. This simulation of the Richtmyer-Meshkov instability, with its extremely fine computational mesh, enables us to study the statistical properties of the turbulence not in some artificial context resulting from arbitrary initial conditions but instead in a physical context realizable in a laboratory experiment. This achievement represents a great advance for our research.

8 Acknowledgments

We acknowledge Terry Heidelberg, Charles Athey, David Fox, James Garlick and Robin Goldstone of LLNL, and David Moffatt and Paul Herb of IBM for their assistance in system administration matters pertaining to the scientific simulation reported here. We acknowledge Roch Archambault of IBM for implementing sPPM-related compiler optimizations, Catherine Crawford of IBM for trouble-shooting last minute problems in the proof-of-principle calculations, and Andrew Wack of IBM for testing, debugging and support for the demonstration runs.

We acknowledge the ASCI program both for its support of the scientific research and for providing the necessary computational resources. This work was performed under the auspices of the U.S.D.O.E. by Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

The University of Minnesota team acknowledges support from the DOE ASCI program, through contracts from both LLNL and LANL, from the DOE Office of Science through grants DE-FG02-87ER25035 and DE-FG02-94ER25207, from the NSF PACI program through subcontracts from NCSA, and from the University of Minnesota's Supercomputing Institute.

References

- [1] S. E. Anderson and P. R. Woodward, World Wide Web <http://www.lcse.umn.edu/research/sppm>, Laboratory for Computational Science and Engineering, University of Minnesota (1995).
- [2] R. D. Richtmyer, Taylor Instability in Shock Acceleration of Compressible Fluids, *Commun. Pure Appl. Math*, 13 (1960), 297; E.E. Meshkov, Instability of the Interface of Two Gases Accelerated by a Shock Wave, *Sov. Fluid Dyn.*, 4 (1969), 101.
- [3] P. Colella and P. R. Woodward, The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations, *J. Comput. Phys.*, 54 (1984), pp. 174-201.
- [4] P. R. Woodward and P. Colella, The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks, *J. Comput. Phys.*, 54 (1984), pp. 115-173.
- [5] B. van Leer, Towards the Ultimate Conservative Difference Scheme. V. A Second-Order Sequel to Godunov's Method, *J. Comput. Phys.*, 32 (1979), pp. 101-136.
- [6] P. R. Woodward, B. K. Edgar and S. E. Anderson, PPMLib, a library of code modules available on the World Wide Web at <http://www.lcse.umn.edu/PPMLib>, Laboratory for Computational Science and Engineering, University of Minnesota (1999).
- [7] Accelerated Strategic Computing Initiative, World Wide Web <http://www.llnl.gov/asci>, Lawrence Livermore National Laboratory Report UCRL-MI-125923.
- [8] M. Vetter and B. Sturtevant, Experiments on the Richtmyer-Meshkov Instability of an Air/SF6 Interface, *Shock Waves*, 4 (1995), pp. 247-252.
- [9] S.E. Anderson and D.H. Porter, Laboratory for Computational Science and Engineering, University of Minnesota, private communication.
- [10] P. E. Dimotakis, The Mixing Transition in Turbulence, *J. Fluid Mech.*, in press (1999).

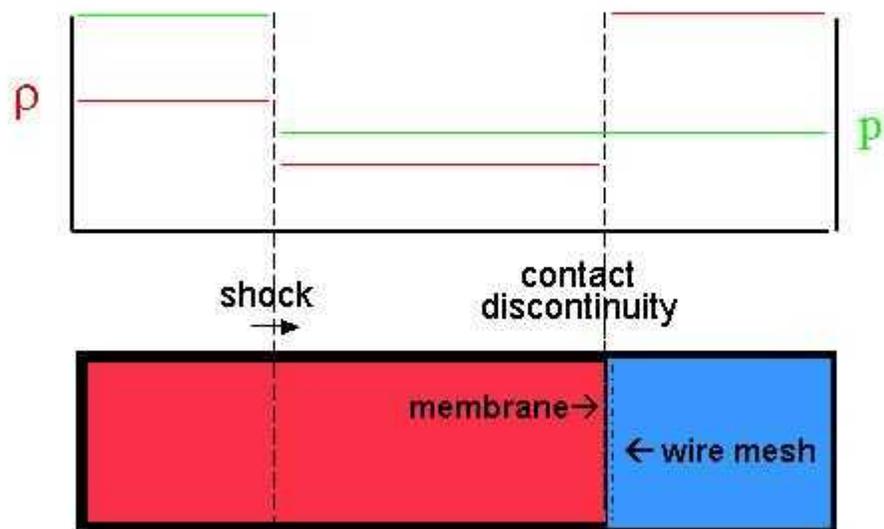


Fig. 1. Diagram depicting shock tube experiment. The two gases are separated by the membrane, which is pushed against the wire mesh. Density is in brown, and pressure in green (not to scale). The shock moves from left to right. The calculation is carried out in a reference frame in which the contact discontinuity is roughly stationary.

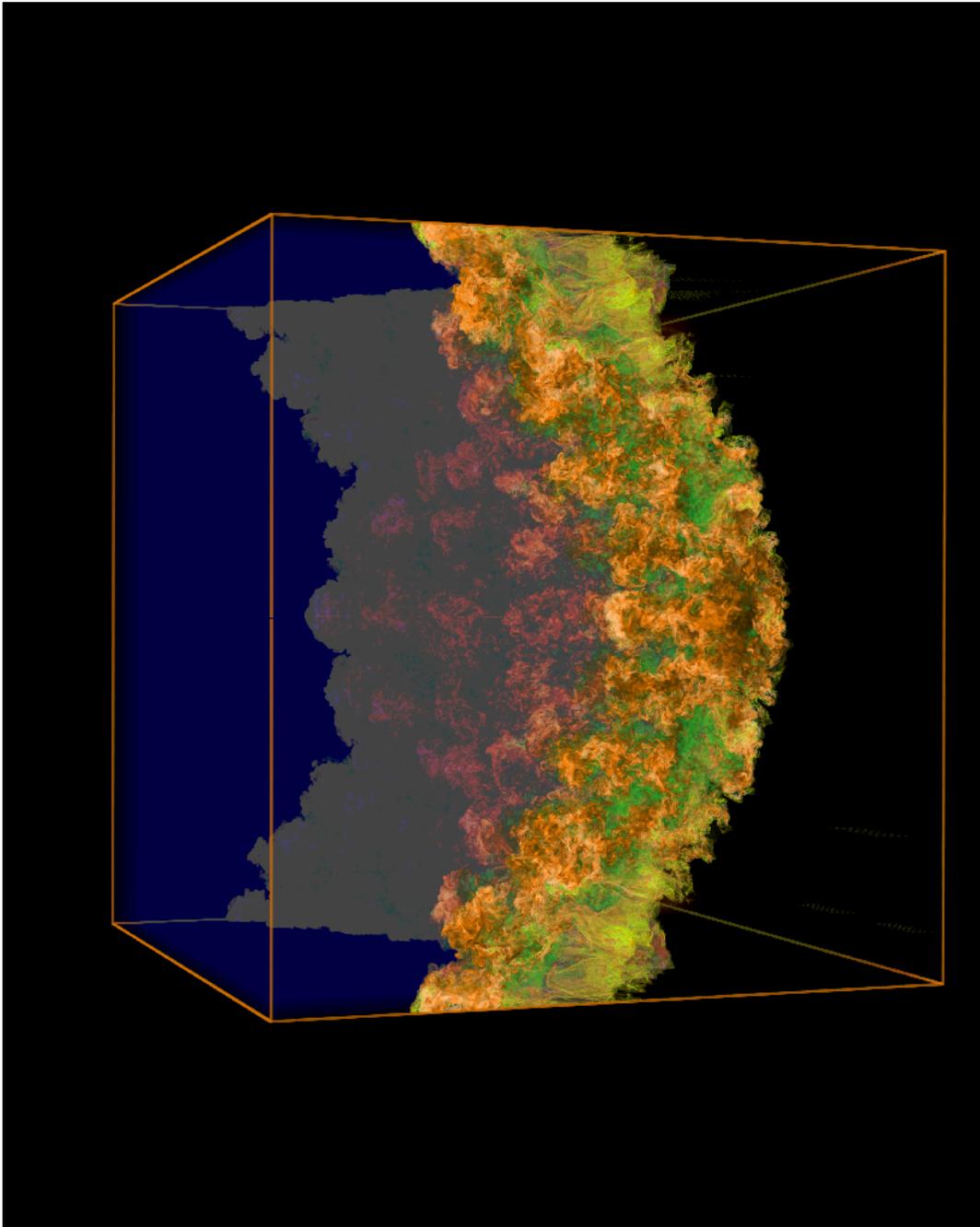


Fig. 2. Volume rendering of entropy for the 2048 x 2048 x 1920 case.

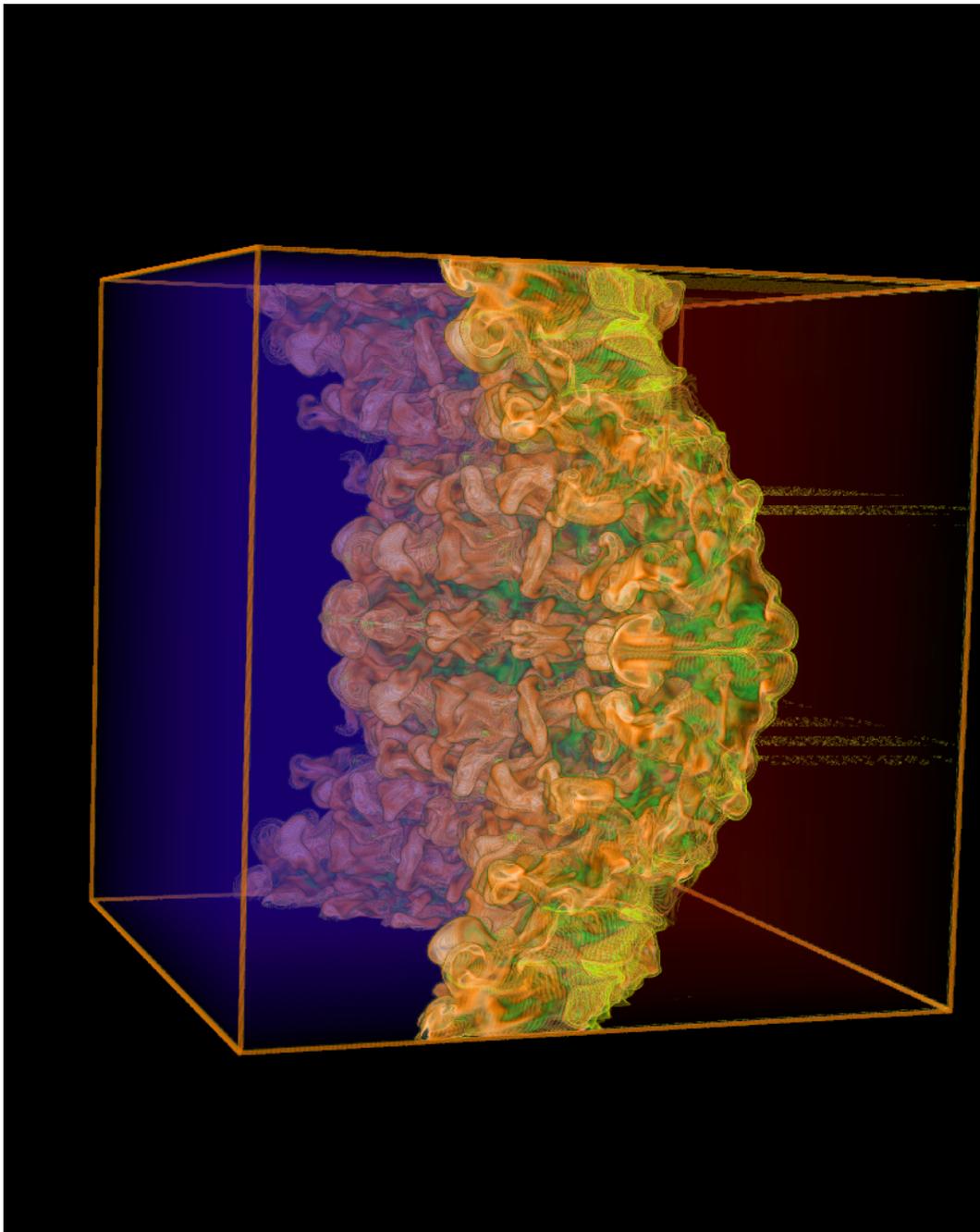


Fig. 3. Volume rendering of entropy for the 384 x 384 x 384 case.

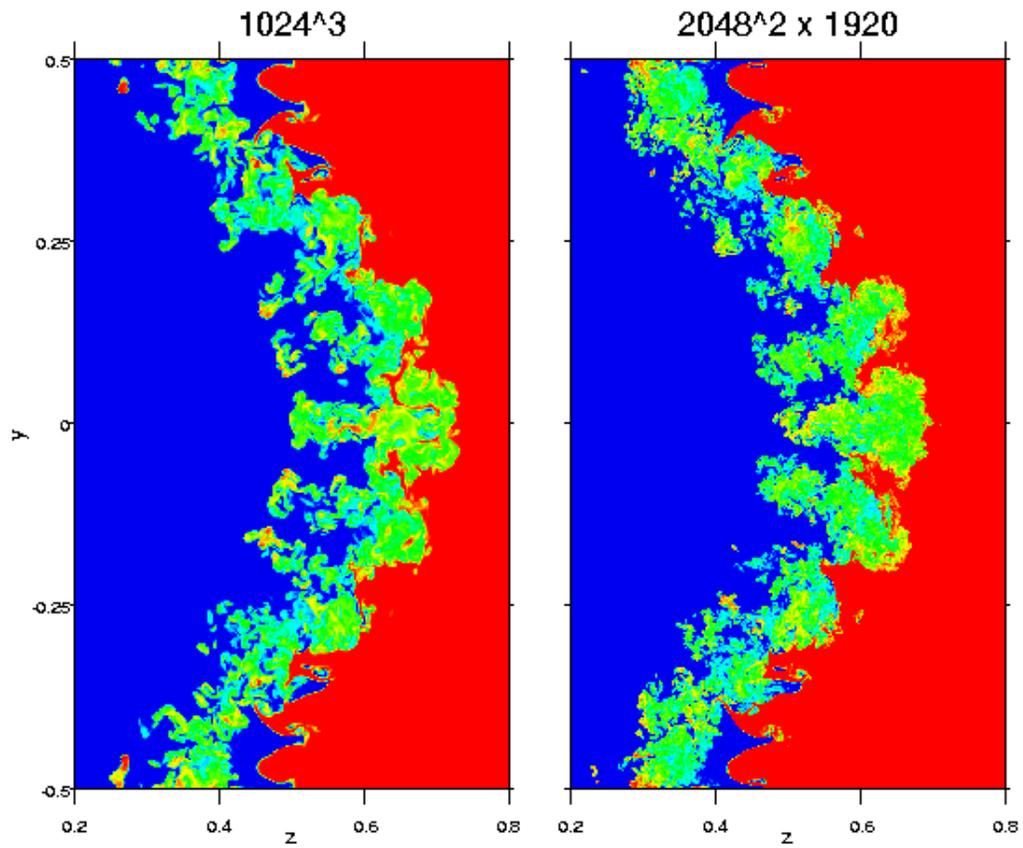


Fig. 4. Comparison of $1024 \times 1024 \times 1024$ case with $2048 \times 2048 \times 1920$ case. Shown is volume-rendered entropy. The fine-scale structure is present in both simulations.

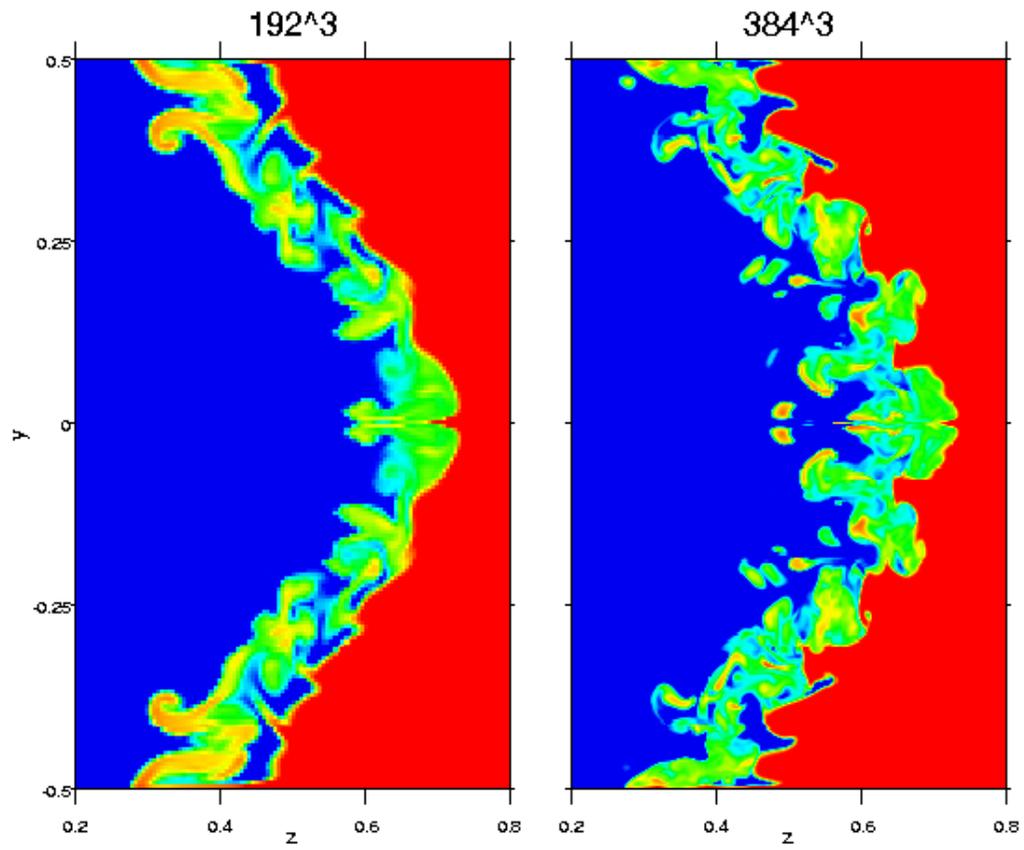


Fig. 5. Comparison of $192 \times 192 \times 192$ case with $384 \times 384 \times 384$ case. Shown is volume-rendered entropy. Both simulations show large-scale structures.

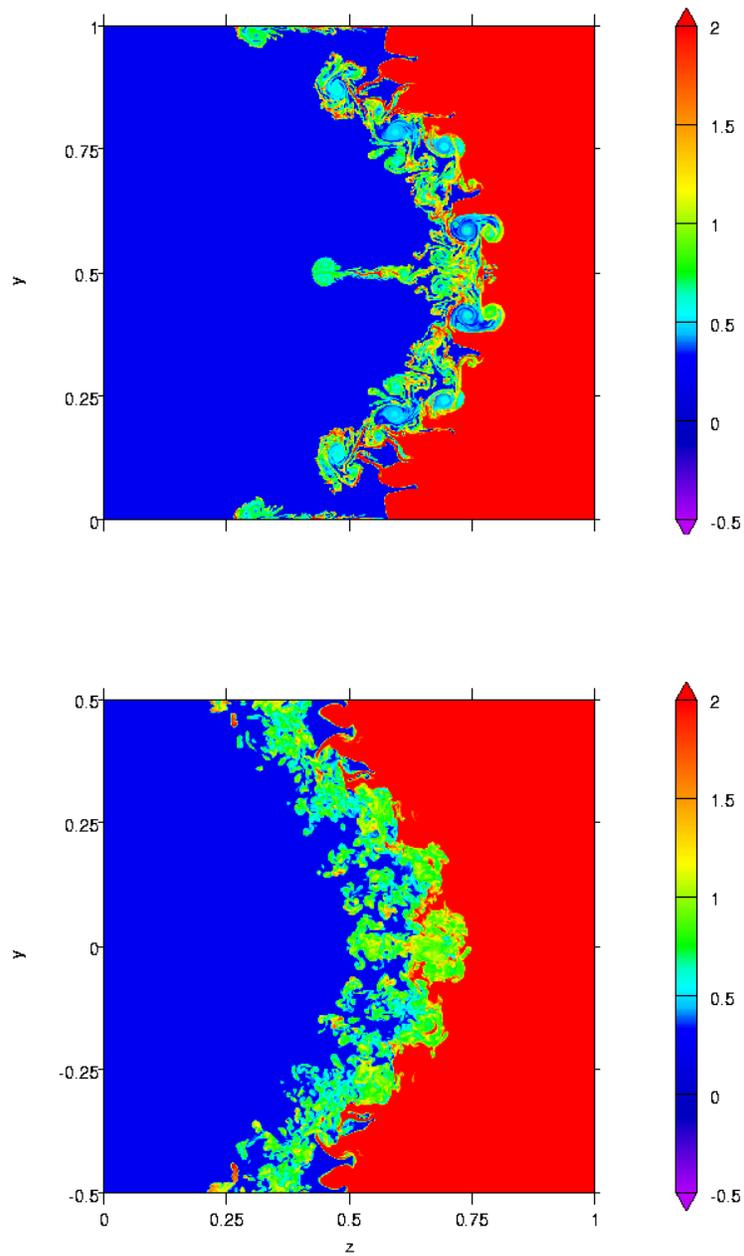


Fig. 6. Comparison of two-dimensional (top) and three-dimensional (bottom) simulations. Each case has 1024 gridpoints per direction. In two dimensions, rather than developing random-looking fine-scale structure, there is an inverse cascade leading to extended structures with sharper boundaries.

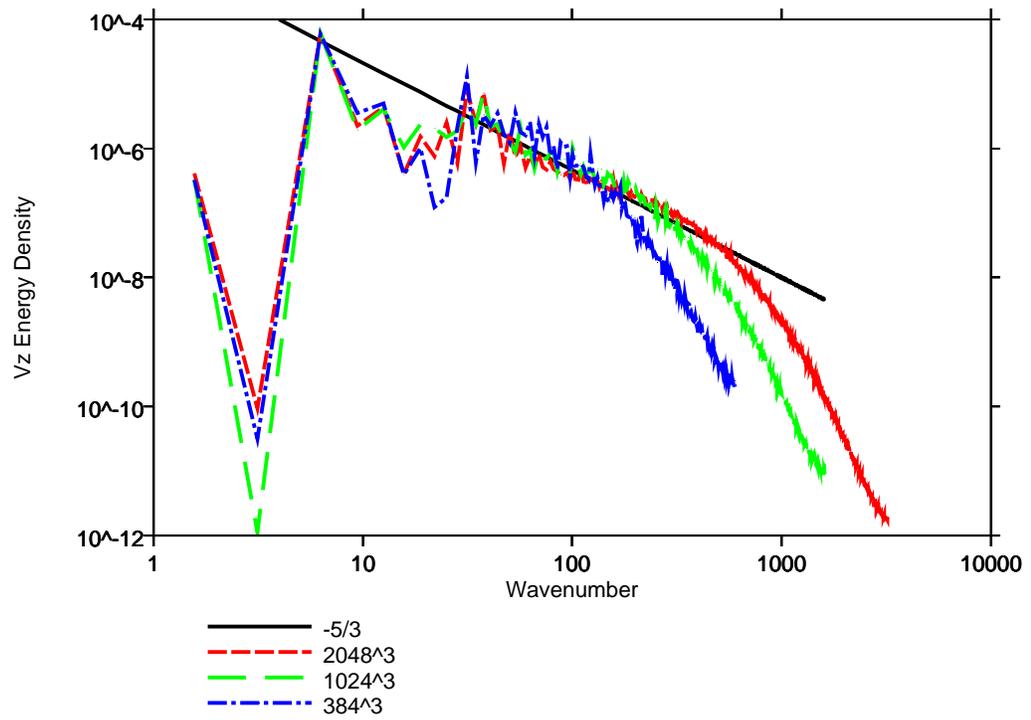


Fig. 7. Energy spectra of z velocity at midplane versus resolution. An inertial range is present, consisting of wave numbers ranging from roughly 30 to 300 for the highest resolution case.

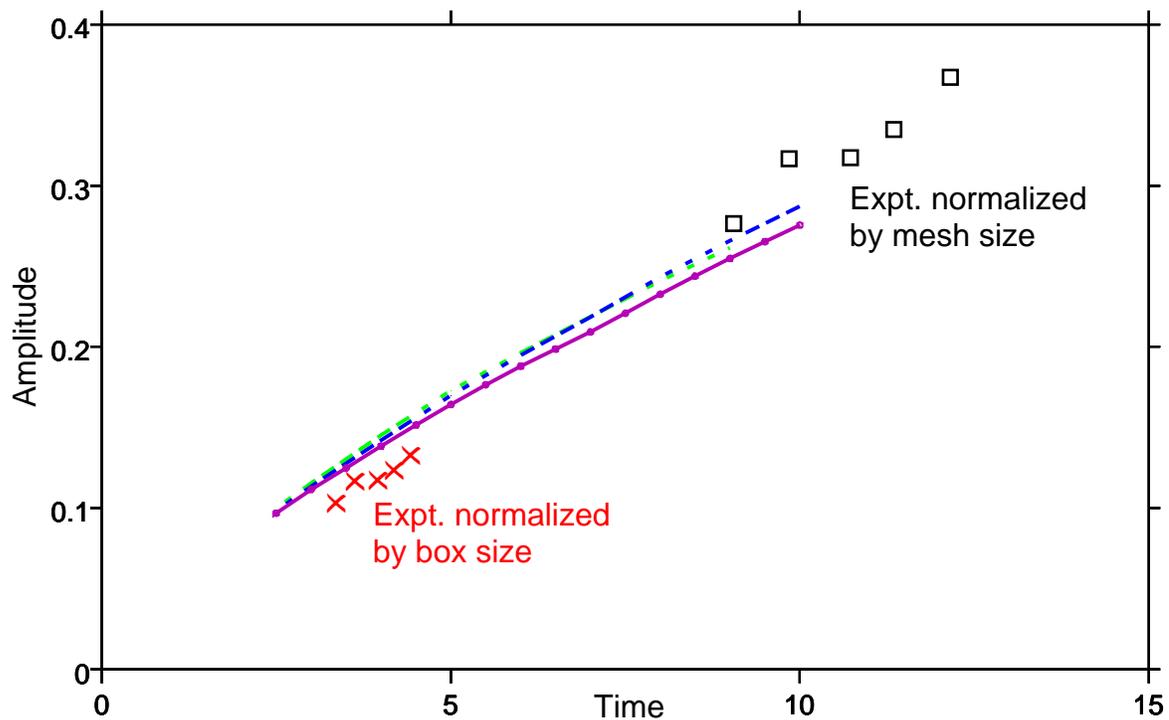


Fig. 8. Mixing layer width at 384 x 384 x 384 (magenta), 1024 x 1024 x 1024 (blue), and 2048 x 2048 x 1920 (green), and compared with experiment. The black boxes correspond to the simulation and experiment having the same wire grid size; the red crosses correspond to the simulation and experiment having the same box size.