Interactive Supercomputing Enabled by Cell Processor Accelerators

Jagan Jayaraj, Pei-Hung Lin, and Paul R. Woodward
Laboratory for Computational Science & Engineering
University of Minnesota

***Abstract*—In the Laboratory for Computational Science & Engineering at the University of Minnesota, our team has used IBM Roadrunner triblades to build an interactive supercomputing system connected to our 12-panel PowerWall display. We have completely redesigned our multifluid hydrodynamics code to run well on this system with fully integrated, prompt fluid flow visualization. Our code is based upon the Piecewise-Parabolic Method (PPM) and the Piecewise-Parabolic Boltzmann scheme (PPB) for moment–conserving advection of the individual fluid mixing fractions [1]. This new code is specially designed for the Roadrunner hardware and has been extensively tested both in our lab at the University of Minnesota and also at Los Alamos. Our approach represents an extreme design in which all computation is performed on the Cell processor accelerator, and the host Opteron is used only for I/O and networking service functions. This strategy avoids frequent data exchanges of small granularity between the host and the accelerator, and any processing stalls that the latency of such exchanges might cause. It also enables a simplified messaging strategy for transport of data between the accelerators. Finally, this approach enables fully asynchronous I/O at essentially no cost in accelerator processing time to the running application. Because our code makes no other use for the host's memory, we are able to place complete copies of the fundamental data for the computation on the host. While the accelerators carry the computation forward, the host can then simultaneously process its copy of the data into compressed formats for disk output or gradually stream its copy out to disk as a restart file. To consolidate such data streams so that only a small number of disk files are generated, we instantiate a small number of specialized MPI processes to receive the individual I/O streams from many hosts and to write them serially out to single disk files. Each such I/O process is assigned to between 32 and 512 worker processes, depending upon the characteristics of the computing system and the particular run underway. Typically, we find that the application computes between 50 and 500 time steps in the interval during which a single time level's fundamental data is dumped to disk in this fashion. Running interactively in our lab, we have one such I/O process streaming compressed data for flow visualization to the 12 workstations attached to our PowerWall display. These 12 workstations use their GPUs to construct volume-rendered images as the computation proceeds. The parameters of the flow visualization are under interactive user control through a graphical user interface (GUI) as the simulation runs. We are using this code in collaboration with colleagues at Los Alamos to study compressible turbulent mixing at accelerated multifluid interfaces. We are also using a separate version of the code to simulate entrainment of stably stratified hydrogen-rich fuel into the convection zone above the helium burning shell in a giant star near the end of its life.**

Keywords – Cell processor, multicore CPU programming, interactive supercomputing, code restructuring.

I. Interactive Supercomputing

Multicore CPUs have enabled us to dramatically improve the delivered performance of our fluid dynamics codes over the last three years. Our work has originally focused on the 9-core IBM Cell processor, which we use in its implementation as an application accelerator in the IBM Roadrunner triblade. However, over the time since the introduction of this groundbreaking CPU, mainstream CPUs such as IBM Power7 and Intel Nehalem-EX have come to offer 8 on-chip cores complete with large on-chip cache memories. The very large speed-ups we achieve with our Cell-targeted code implementation, which are factors of about 40 for our codes when compared to the single-core Intel Pentium CPUs that we used in 2006, are therefore now available on less exotic and more easily programmed hardware. We have exploited the Cell processor's unprecedented computational power to build a modest system in our lab that is capable of running significant fluid dynamics problems at fully interactive speeds. Our initial system for this work consisted of 6 Roadrunner triblades from IBM, containing 24 Cell processors with 6 host AMD Opteron CPUs on an Infiniband (IB) network fabric. In the last year, we have added to this system an Intel-based system containing roughly the same number of processing cores but running our new codes at about the same to 2.5 times the speed of the Cell-based system, depending upon the application and the accuracy desired. Although we have achieved our original goal for the system – namely interactive execution of fluid flow simulations at about one teraflop/s sustained – we plan to enhance the Intel-based system with GPU accelerators to bring the performance up by an additional substantial factor. In this paper, we will describe the system and how we have restructured our codes to exploit it.

II. Necessary Code Modifications

The largest change in our codes that has made interactive supercomputing possible is a fundamental restructuring to permit a streaming model of computation in which small, indivisible data records are (1) prefetched from memory, (2) unpacked in the on-chip cache into separate arrays representing values of different fluid state variables, (3) processed in a fully

This work has been supported through grant CNS-0708822 from the National Science Foundation, and our code development has been supported by the Department of Energy through a contract from the Los Alamos National Laboratory. We also acknowledge support to our interactive supercomputing project from the Minnesota Supercomputing Institute.

vectorized mode using 4-way SIMD primitive operations, (4) repacked into data records, and (5) written back into memory. The code implementation techniques involved are described elsewhere [2-4]. These transformations required rewriting every line of our codes in this new fashion. We are now building automated code translators to reduce this programming burden for ourselves and for others. These code transformations deliver the performance increase needed for interactive execution on our modest system in the Laboratory for Computational Science & Engineering (LCSE). However, this is not all we need to do. To interactively control the simulation and the visualization of the developing fluid flow, we need to collapse our data analysis and visualization pipeline into just a small number of stages. Much of the processing that we used to do separately on individual workstation systems now must proceed in a highly parallel fashion exploiting the Cell or other multicore CPU accelerators of the simulation code. We have therefore implemented inside the code itself the generation of bricks of bytes representing various quantities that we wish to view. We do this while the fundamental flow state variables are in on-chip storage in step 3 of the above 5-step data update procedure. This avoids the largest cost of the data preparation process, which is simply reading the data from memory, or, worse, from disk. This visualization data is sent via MPI messages to the I/O process for the run, which consolidates it and forwards it via specialized socket code to a control workstation with a large RAM disk. Here the bricks of bytes are reformatted into a blocked octree structure and then broadcast to the 12 workstations attached to the 12-panel PowerWall display. These 12 workstations collaboratively construct volume rendered images of the developing flow, with the viewing and rendering parameters under interactive user control. We find that this entire process of data preparation, dispatch, reformatting, broadcast, and rendering takes about 10 seconds. During the 10-second interval before the next updated visualization data arrives, the PowerWall system can generate many views of the present data. Thus we can navigate through the data while it is periodically updated to reflect the developing flow.

III. Modes of Use

Our original intent in using this interactive supercomputing system was to be able to explore parameter spaces of particular flow problems, quickly locating parameter values of interest for much more large-scale simulations that we would run on much larger systems. We have used the system extensively in this mode over the last couple of years. Our graphics software can deliver the flow visualizations to most any display, so we are not always in front of the PowerWall when we are doing these interactive runs. The visualizations and the runs can be controlled from a laptop or desktop machine anywhere on the Internet (with appropriate permissions of course), and this capability has proven to be immensely useful. Another use for the system's interactive simulation capability was not our original focus, but has proven equally important. This other

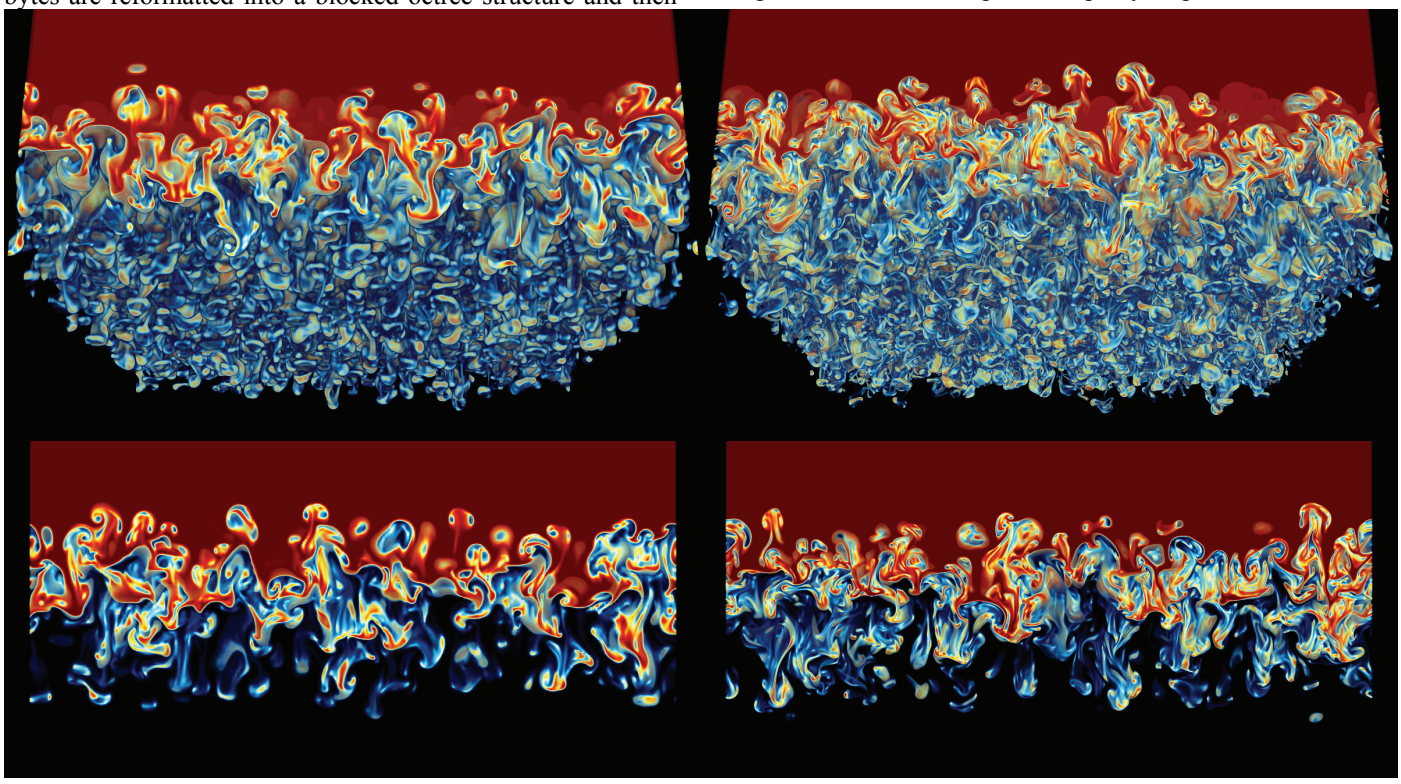

Figure 1. Two runs of a multimode Rayleigh-Taylor instability problem are here compared through volume renderings of the multifluid mixing fraction near the unstable interface. We see the advancing spikes from below in the upper view, and we see a thin section of the flow in the lower view. At the left, we use a grid of $256^2 \times 768$ and at the right we use $512^2 \times 1536$ cells. The simulations ran in the LCSE at 1.04 and 1.17 Tflop/s sustained (5.4 and 6.1 Gflop/s/core, or 23% and 26% of peak in 32-bit floating point mode), but the run on the left took 33 minutes, while that on the right took 8 hours and 18 minutes. The two results for this chaotic flow differ, in a statistical sense, only in fine detail, while the penetration of the mixing layer and the character of the flow is essentially the same in each. Thus it is clear that the run on the left gives coarser but still very useful results rapidly enough to enable full user interaction with the run.

use is for algorithm and code development. It has proven to be invaluable to be able to carry out interactive runs on demand and very quickly in order to test new computational algorithms. These tests can also have the character of parameter space exploration, because one wishes to test the behavior of a new technique in a wide variety of circumstances. One might think that such tests can always be small, but this is not the case when the code is ultimately to be used at a scale of as many as 100,000 processor cores. Some of our tests aimed at the elimination of algorithmically generated glitches in multifluid interface representations had to be taken up to grid sizes as large as $256^2 \times 2048$, which can still be run interactively on our LCSE system.

IV. The Trade-off between Accuracy and Computation Speed

The usefulness of interactive simulation depends entirely upon a trade-off between the accuracy of the flow representation and the speed of the computation. We will try to quantify this a bit for our system through examples from our recent research collaborations. The first example is a collaborative study of compressible, turbulent mixing by the Rayleigh-Taylor instability. We are collaborating with Guy Dimonte, Chris Fryer, William Dai, and Gabe Rockefeller at Los Alamos National Laboratory in this work. The second example is a study of giant stars in late stages of their evolution, when ingestion of unprocessed gas into convection zone above the helium burning shell can lead to a hydrogen ingestion flash. In both of these examples the flow Mach numbers are low, so that we must take a great many time steps to advance the flow appreciably. Aside, of course, from making the computing system and/or the code run faster – a strategy that we have already discussed – we can speed the simulation up by a combination of the following approaches: (1) reducing the grid resolution, (2) reducing the amount of the flow domain contained in our simulation grid, or (3) modifying the flow parameters to make the Mach numbers larger. The fidelity of the resulting simulation to the flow we wish to understand will be reduced, but the simulation time will also be reduced. The usefulness of the simulation will depend upon the degree of fidelity that we are able to retain while making the simulation interactive. In our work we have mostly used techniques 1 and 2, and have only experimented in a limited fashion with technique 3. Examples, with flow visualizations and simulation running times, will be presented at the conference. One example using techniques 1 and 2 together for exploration of multimode Rayleigh-Taylor flow problems is shown in Fig. 1.

V. Future Work

We are building automated code translators using the ANTLR framework to help reduce the programming burden associated with the techniques we have found so helpful in accelerating our codes on the Cell processor. Our present translator converts a highly specialized and very high performance Fortran expression into C with vector intrinsics for Cell. We are building a variant of this tool to convert this Fortran into Fortran with SSE or Altivec intrinsics for Intel, AMD, and IBM standard multicore CPUs. Benefits are roughly factors of 4 in code execution speed. At this writing, only the Intel 9.1 Fortran compiler (or earlier) will insert these intrinsics automatically for our code expression. We are also building a related translator to produce CUDA code from our special Fortran expression. All these translations produce code that is ridiculously difficult to write and maintain from high level code that is extremely difficult to write and maintain. Therefore, to lower the programming burden still further, we are building code translators to produce our special Fortran expression from more easily written Fortran code. We are developing the input Fortran rules inspired by the particularly readable and maintainable expressions used for meteorological codes by Robert Wilhelmson and his collaborators at the University of Illinois. We call this Wilhelmson Fortran. Such codes consist of a large number of bursts of computation expressed as individual doubly or triply nested loops running over an entire computational domain for a network node. According to the transformations described in [2-6], our translator will in-line, align, fuse, and restructure working memory, under programmer directive control, to accelerate the code execution. These transformations involve, critically, restructuring of the data layout in node main memory, as specified by the programmer via values of Fortran code parameters and by directives to the translator tool. Therefore one cannot expect these code improvements to be accomplished by any standard Fortran compiler. Performance benefits over Wilhelmson Fortran expressions for codes of our own that we have tested range from factors of 3 to 6 on our quadcore-Intel-Nehalem-based cluster. We will be working over the next year with our Los Alamos collaborators and with Wilhelmson and his team to identify the code changes programmers may be expected to make in their Wilhelmson Fortran expressions that will enable useful translators to be built with our small team in a reasonable length of time. We also intend to explore the effectiveness of these code transformations in the more challenging contexts of adaptive mesh refinement (AMR) codes.

References

[1] P. R. Woodward et al. The Piecewise-Parabolic Boltzmann Advection Scheme (PPB) Applied to Multifluid Hydrodynamics, Los Alamos National Laboratory Report LA-UR 10-01823, March, 2010.

[2] P. R. Woodward, J. Jayaraj, P.-H. Lin, and P.-C. Yew, Moving scientific codes to multicore microprocessor CPUs, *Computing in Science and Engineering*, Nov. 2008, 16-25.

[3] P. R. Woodward, J. Jayaraj, P.-H. Lin, and W. Dai, First experience of compressible gas dynamics simulation on the Los Alamos Roadrunner machine. *Concurrency and Computation: Practice and Experience*, 2009.

[4] P. R. Woodward, J. Jayaraj, P.-H. Lin, and D. Porter, Programming techniques for moving scientific simulation codes to Roadrunner. Tutorial given 3/12/09 at Los Alamos. Available at www.lanl.gov/roadrunner/rrtechnicalseminars2008.

[5] P.-H. Lin, J. Jayaraj, and P. R. Woodward. A strategy for automatically generating high performance CUDA code for a GPU accelerator from a simplified input code expression. Extended abstract submitted to SAAHPC 2010, May, 2010; accepted for poster presentation.

[6] P. R. Woodward, J. Jayaraj, P.-H. Lin, P.-C. Yew, M. Knox, J. Greensky, A. Nowatski, and K. Stoffels. Boosting the Performance of Computational Fluid Dynamics Codes for Interactive Supercomputing, Proc. International Conf. on Computational Science ICCS 2010, Amsterdam, May, 2010, in press.