MP ZEUS: testing and performance analysis of the message passing ZEUS fluid simulation package

Asif ud-Doula and Hans-Reinhard Müller
Department of Physics
and
Samuel Kalet and Joe Pellegrino
Department of Computer and Information Sciences
University of Delaware

ZEUS, developed by the Laboratory for Computational Astrophysics (LCA) of the National Center for Supercomputing Applications, has become a standard tool in the astrophysical community for a wide range of fluid dynamics simulations. To solve more complex problems with magnetic fields or self-gravity and to follow the evolution of these problems over long periods of physical time, many times the computation time of other problems is required. To ease these computation time requirements, LCA released parallel version of the sequential ZEUS code called ZEUS-MP: Multi-Physics, Massively-Parallel, Message-Passing code.

This paper first verifies the correctness of ZEUS-MP. The performance of ZEUS-MP is then evaluated on two clusters of different architectures using a problem that is bundled with the ZEUS-MP package. Performance is found to be significantly less than ideal on both Intel and SPARC-based clusters, which is likely rooted in the lack of architecture-specific optimizations for architectures other than SGI.

1. INTRODUCTION

ZEUS has become a popular tool in the astrophysical community for a wide range of fluid dynamics and magneto-hydrodynamics (MHD) simulations. It has been developed over many years by the Laboratory for Computational Astrophysics (LCA) of National Center for Supercomputing Applications (NCSA). ZEUS has been used to simulate supernova explosions, gravitational collapse, study solar wind, or even star formation, and for many other astrophysical problems.

Many of these problems require fine grids and consequently significant computing time. But as the problems become more complex, for example if magnetic field or self-gravity is included, the simulation times increase many folds. Also, astrophysicists very often wish to follow the evolution of their problem over long periods of physical time which usually corresponds to a large number of iterations. To ease this problem LCA released a parallel version of the sequential ZEUS code called ZEUS-MP where the MP stands for Multi-Physics, Massively-Parallel, Message-Passing code. The current working release is 1.0.

2. MOTIVATION

This paper first examines ZEUS-MP for its performance and correctness. In order to develop confidence in any numerical code, one has to do some tests the result of which we know analytically, and make sure that the code gives reasonable results. Secondly, this paper determines whether ZEUS-MP yields time savings that makes

it worthwhile to use ZEUS-MP as opposed to the sequential implementation of ZEUS. Finally, ZEUS-MP is evaluated for its performance on clusters of different architectures. A Linux Beowulf cluster is given special attention due to its growing popularity in academic settings.

3. PROBLEM STATEMENT

This evaluation considers the ZEUS-MP package with respect to compilation, configuration, execution, performance, portability, and scalability.

3.1 Correctness

Running a short test problem and comparing the results to those of the ZEUS-MP running on one processor without using MPI allowed the correctness of ZEUS-MP to be confirmed. An isothermal solar wind problem was used for this testing. A detailed description of this isothermal solar wind problem can be found in the Test Problem Descriptions section.

3.2 Analysis of program structure

Parts of the source code of ZEUS-MP were analyzed to determine the techniques that it uses to achieve parallelization. The details of the domain decompositition that ZEUS-MP uses to distribute the problem's data and the ways that it uses MPI were studied with the most attention in this analysis.

3.3 Scalability

By varying the number of processors used for parallel computation the scalability of the ZEUS-MP was studied. Speedup and efficiency were also calculated to put timings into context.

3.4 Portability to different clusters

Running ZEUS-MP on two different clusters allowed for understanding how easily the source code could be ported. Configuration and hardware differences between clusters allowed the testing of different aspects of performance and scalability. One of the clusters utilized was a Linux Beowulf cluster composed of 16 nodes, each with a 400MHz Intel Celeron processor. The second cluster was composed of 20 Sun Ultra Enterprise 450s, each with 4 250MHz SPARC Ultra-II processors and 512MB of physical memory.

4. ANALYSIS OF PROBLEM

Since ZEUS-MP has been released for public use, we expected that it would yield correct results. Benchmarks on the ZEUS-MP web site and in Norman [1999] suggest that ZEUS-MP should show a near-linear speedup, and we expected to observe the same trend in our own benchmarks. We also expected that the code would port easily so that benchmarks could be collected for several clusters.

5. TEST PROBLEM DESCRIPTIONS

5.1 Supersonic Jet

In the initial state, the three-dimensional (3D) simulation grid is filled with a uniform density (of normalized value 1) and uniform pressure (also 1). This ambient

medium is non-moving. At $t = 0$, a supersonic, hydrodynamic jet is introduced into the simulation. It is realized as inflow through a circular orifice in the center of the $x = x_{min}$ boundary plane. The orifice is 1 length unit in diameter, whereas the physical dimensions of the $y - z$ plane are 15×15. The circular shape of the inflow boundary is approximated in Cartesian coordinates. The jet streams with Mach 6.0 into the ambient medium, with one tenth of the density of the ambient medium (at ten times the temperature, such that the pressures are equal).

It takes the jet shock front about 8 time units to traverse the length of the simulation grid. Snapshots of the fluid density, taken at intervals of 2.5 time units, are shown in Figure 1. While the simulation boundaries are outflow at $x = x_{max}$, and periodic in y and z directions in Figure 1, many of the timing studies below have been carried out with outflow boundary conditions also at y_{min}, y_{max}, z_{min}, and z_{max}. In certain domain decompositions, these boundary conditions ensure that no extra communications are necessary to carry out the periodic boundary conditions.

We operate the jet problem on a relatively small grid ($128 \times 32 \times 32$), and also on an almost eight times bigger grid ($248 \times 58 \times 58$). Different ways of domain decomposition are studied below.

5.2 Sedov Blast Wave

Like in the jet problem, a uniform medium is seeded with a discontinuous perturbation. In contrast to the jet, the perturbation (a small sphere of 100-fold density and million-fold pressure) is in the interior of the 3D simulation grid, and is allowed to evolve. Fluid is allowed to flow out through all Cartesian boundaries. In further contrast to the jet problem, the blast is a MHD problem, where the initial field is uniform along the x axis and breaks the symmetry, resulting in an ellipsoidal perturbation when time evolves. Figure 2 illustrates a blast wave.

The fast shock expansion requires only simulations short in physical time. The simulations were performed on a $64 \times 64 \times 64$ grid.

5.3 Isothermal solar wind

The corona of the sun is very hot in relation to its surroundings, in fact it can reach several millions degrees Kelvin. Due to this large temperature and density stratification, there is a significant pressure gradient present near the surface of the sun and this pressure gradient causes an outflow of matter from the sun into its surroundings known as the solar wind, predicted by Eugene Parker in 1958, and first discovered in 1960. This is a purely hydrodynamics (HD) simulation represented in spherical coordinates in one dimension and runs should take no more than a few minutes assuming about 400 grid points equally spaced. We expect that this problem will take longer running in ZEUS-MP because its serial runtimes are very small. This problem is computationally easy.

6. STRATEGY FOR PARALLELIZATION

ZEUS-MP is an SPMD parallel code that utilizes the MPI message-passing library for communication between processors. ZEUS-MP uses domain decomposition to distribute the data from the grid of the simulation to the processors involved in the parallel computation. Since ZEUS-MP is inherently 3D, this domain decom-

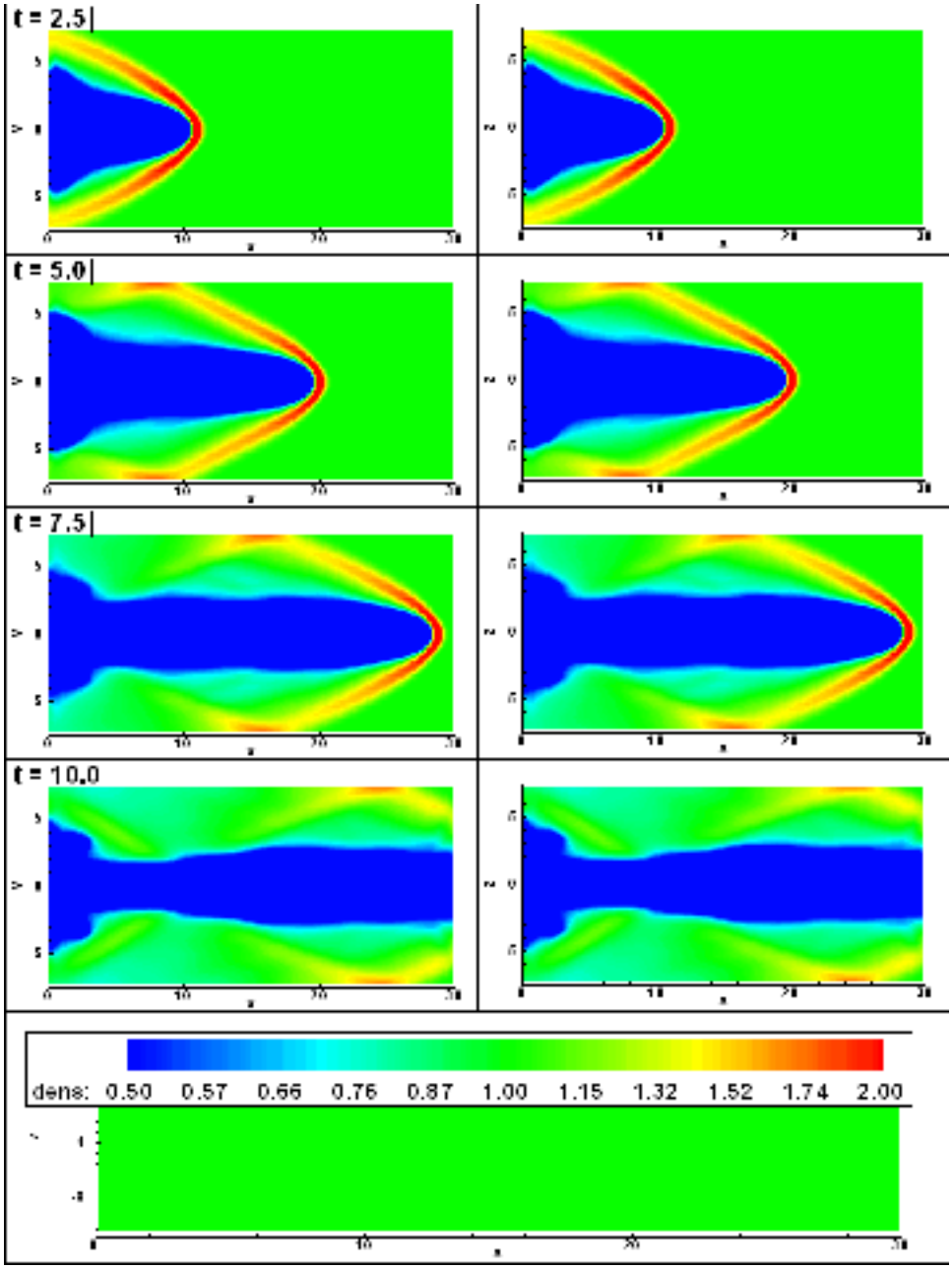

Fig. 1. Supersonic Jet

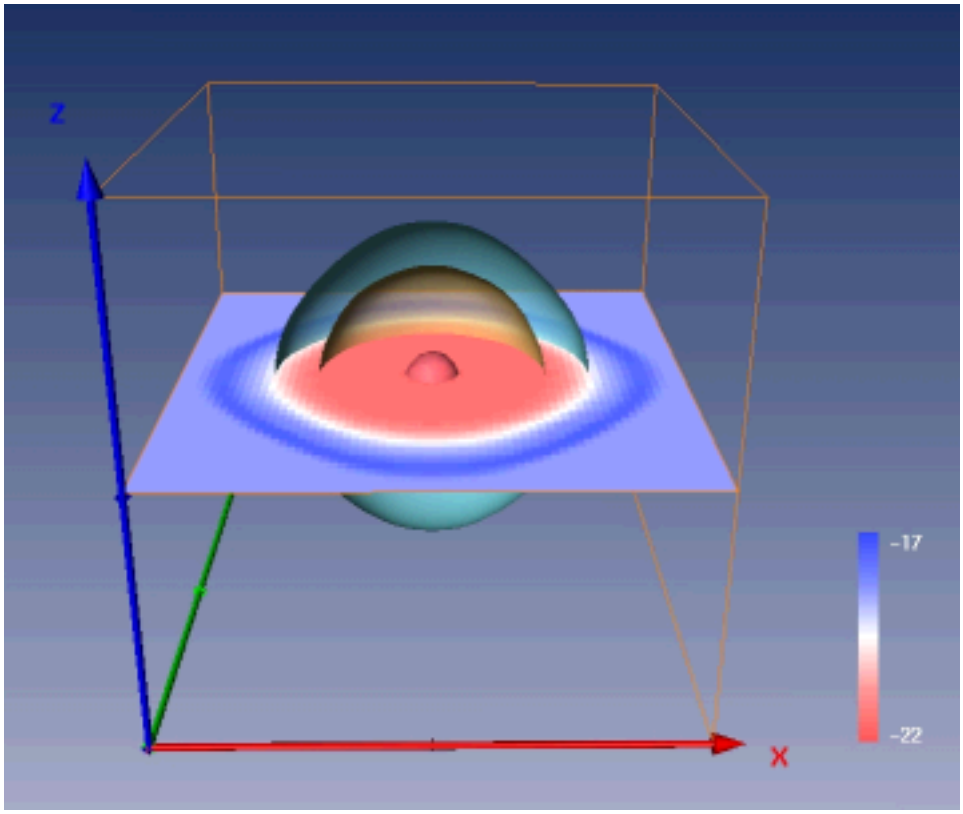

Fig. 2. Sedov Blast Wave

position involves partitioning the data into cubes and distributing this data to the assigned processor once at the beginning of execution. These cubes may be distorted from perfect cubes, but will always have three dimensions. Each processor's data includes two additional layers of data on all of its 6 boundaries, known as ghost zones. Ghost zones serve as boundaries for the grid. Boundaries that are physical (exterior) boundaries of the domain are handled as such. Interior boundaries are non-existent in the serial ZEUS version, but appear in ZEUS-MP due to the domain decomposition. The last two active zones of one processor are always communicated to the ghost zones of the neighboring processor. In this way, boundary values are shared between processors with MPI communication routines throughout the duration of a simulation.

The geometry of the cubes is user-determined at runtime. There are directives in the `mpitop` section of the runtime control file (`zmp_inp`) that control the geometry of the domain decomposition for the simulation. They are named `ntiles(1)`, `ntiles(2)`, and `ntiles(3)`, each representing one of the dimensions. Figures 3, 4 and 5 offer graphical representations of domain decompositions that were used in our performance studies and their associated `ntiles` values in a Cartesian coordinate system.

Fig. 3. Domain Decomposition with ntiles(1)=8, ntiles(2)=1, ntiles(3)=1

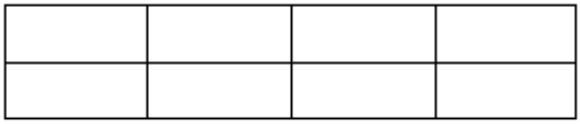

Fig. 4. Domain Decomposition with ntiles(1)=4, ntiles(2)=2, ntiles(3)=1

The selection of geometry for the domain decomposition can greatly impact the runtime of a simulation since ZEUS-MP must coordinate communication of boundary values between neighboring processors. In general, minimizing the total surface area on boundaries between processors will also minimize the number of values that must be communicated between processors, potentially reducing runtimes. However, this greatly depends on the type of problem that ZEUS-MP is simulating during an execution. If the problem is periodic along any boundary the edges defining that boundary in the domain will have to be communicated to the opposing edge of the domain, introducing additional communication costs that are not captured in the general surface area observation. Figure 6 illustrates a problem that is periodic in x. The four processors communicate values along their boundaries as indicated by the arrows and also across their edge boundaries in x, since this problem is periodic in x.

7. IMPLEMENTATION DETAILS

The initial decomposition of the domain across all processors participating in a simulation is handled by the code in the subroutine `mstart.F` that handles the bookkeeping of the domain decomposition. Below is an excerpt of the code from this source file `mstart.F` that defines the initial domain decomposition. Much of the work is done by the `MPI_CART_CREATE` procedure, which accepts the topology (defined by the `ntiles` values in `zmp_inp`), periodicity for each dimension (defined by the `periodic` values in `zmp_inp` and creates a new Cartesian communicator that arranges the processors in the topology to allow for the most efficient communication. This relies heavily on knowledge that the MPI implementation has of the cluster's hardware, especially costs associated with transferring communicated data across the network that connects the processors. The calls to `MPI_CART_SHIFT`

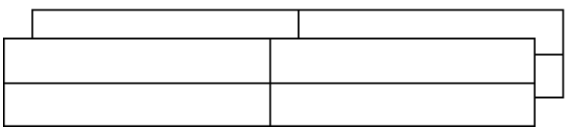

Fig. 5. Domain Decomposition with ntiles(1)=2, ntiles(2)=2, ntiles(3)=2

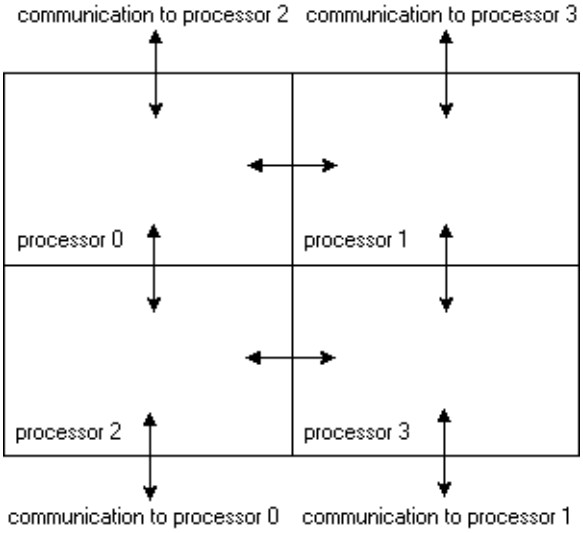

Fig. 6. Additional communication introduced in periodic problems

allow the node to locate the rank of its neighbors in the Cartesian communicator created by `MPI_CART_CREATE`. The call to `MPI_CART_COORDS` allows the node to find its coordinates within the same Cartesian communicator.

```
  c
c Create a virtual Cartesian topology for the domain decomposition.
c
call MPI_CART_CREATE( MPI_COMM_WORLD, 3, ntiles, periodic
& , reorder, comm3d, ierr )
call MPI_COMM_RANK( comm3d, myid, ierr )
call MPI_COMM_SIZE( comm3d, nprocs, ierr )
c
c Find the ranks of my neighbors; find my virtual Cartesian coords.
c
call MPI_CART_SHIFT( comm3d, 0, 1, n1m, n1p, ierr )
call MPI_CART_SHIFT( comm3d, 1, 1, n2m, n2p, ierr )
call MPI_CART_SHIFT( comm3d, 2, 1, n3m, n3p, ierr )
c
call MPI_CART_COORDS( comm3d, myid, 3, coords, ierr )
```

Once the domain decomposition is complete the real calculation begins. At each timestep, communication between tiles is initiated along one particular axis. Once this is complete computation along that axis is performed. To ensure that all the processors remain synchronized, calls to `MPI_BARRIER` are used. Subsequently, calculations along the remaining two axes are performed. An alternate method to achieve the same result could be to communicate between tiles along all three axes, then perform the numerical calculations. The authors of ZEUS-MP found that

this was not as efficient as the method actually used. Another important step in simulation is calculating the timestep, dt. One has to make sure that all the tiles use the same dt. Each tile calls a subroutine called `nudt` to calculate dt, which uses a Courant condition to find the next timestep. `MPI_ALL_REDUCE` is then used to find the minimum of all these timesteps. This new minimum dt is then communicated to all the processors, and a new iteration can then begin [Fiedler 1997].

8. TESTING AND PERFORMANCE

8.1 Correctness

Correctness was tested by using the solar wind problem in ZEUS-MP first running one one processor without using MPI and then on four processors. ZEUS-MP running one one processor does not involve the complexities of domain decomposition and boundary conditions between tiles. The results of these two runs are plotted in Figure 7. The results from the two runs are overlayed, confirming that the results from the two runs are identical. Additionally, simulations on multiple processors using MPI yielded the same results even as the number of processors and domain decomposition guidelines were changed.

8.2 Scalability

Scaling timings from runs of the jet hydrodynamics simulation with a (relatively small) grid size of $128 \times 32 \times 32$ grid points with y and z periodic on a Linux Beowulf cluster are illustrated by Figure 8. The speedup derived from the timings of Figure 8 are presented in Figure 9. The runtimes and speedup do not show a linear speedup as suggested by the ideal line on each plot. This is much different than the observations made in Norman [1999] where near-ideal scaling is found.

These near-ideal scalings were observed in an environment that is significantly different from the results presented here. The near-ideal results were found with simulations ranging up to 256 processors for grid sizes of up to 512^3 grid points, making those problems much larger and solved by many more processors than used for the study presented in this paper. The near-ideal benchmarks were also performed on an SGI Origin 2000 with presumably very fast inter-processor communications. SGI is the preferred target architecture for ZEUS-MP since optimizations in the source code have been tailored specifically for this architecture. Loop optimizations including data prefetching are used to increase cache performance [Fiedler 1997].

The scaling trends observed running the same jet hydrodynamics simulation on the Linux Beowulf and the Sun clusters were very similar. Speedup observed on the Sun cluster is presented in Figure 11. Memory limitations prevented us from obtaining a single processor timing for the Sun cluster. The single processor baseline for calculating speedup was estimated by dividing the time for 2 processors by 2. This leads to an ideal speedup at 2 processors, which is artificially introduced by this estimation. The 16 processor run is also inconsistent.

8.3 Geometric Effects

Selection of different topologies for the distribution of the domain to each processor can have significant impact on the runtime of a simulation on ZEUS-MP. Table

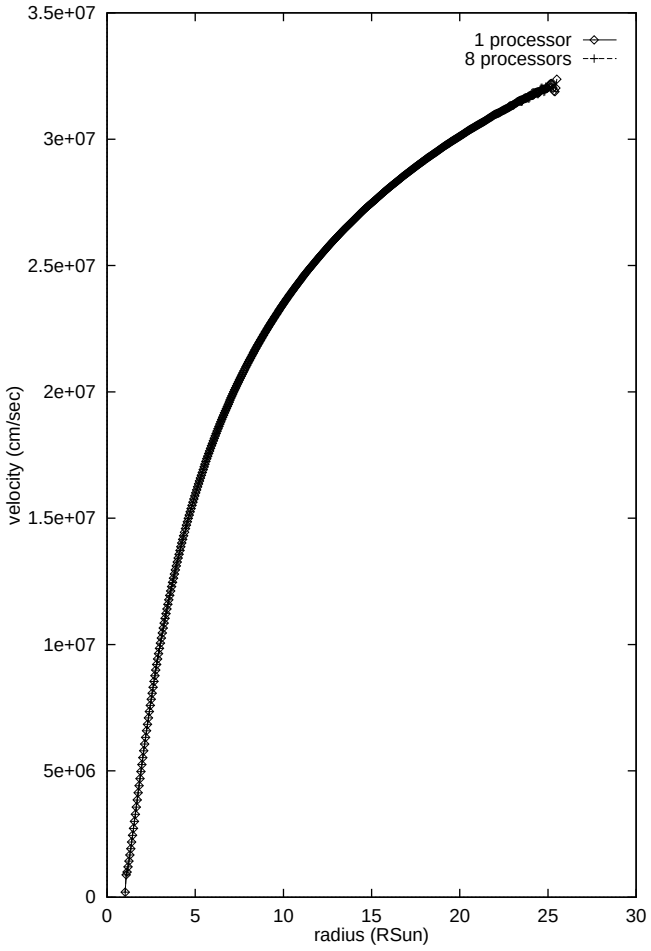

Fig. 7. Two runs of the solar wind problem in ZEUS-MP overlayed to illustrate correctness

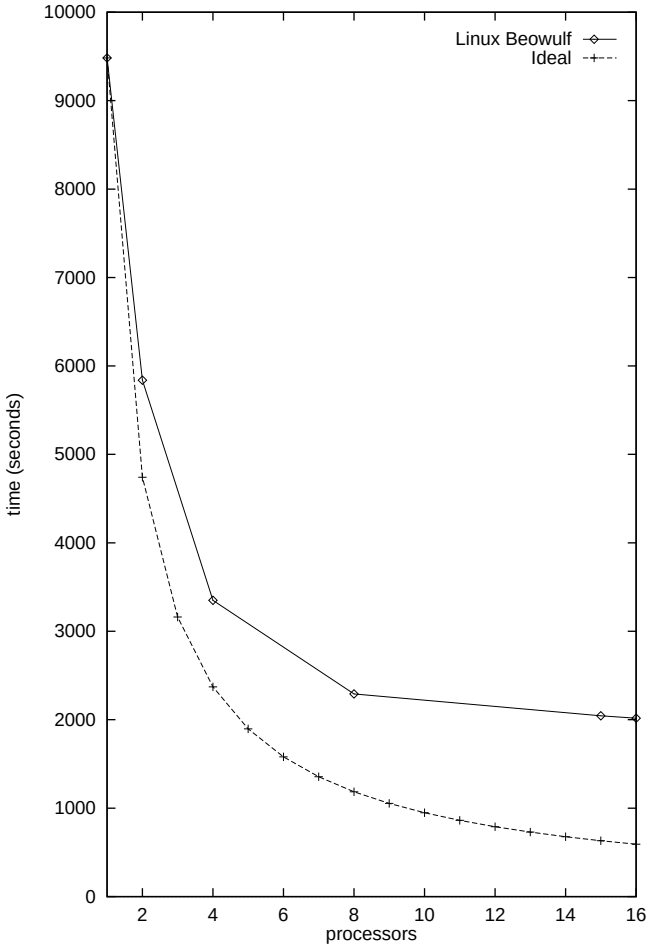

Fig. 8. Scaling for a jet problem (Intel Linux)

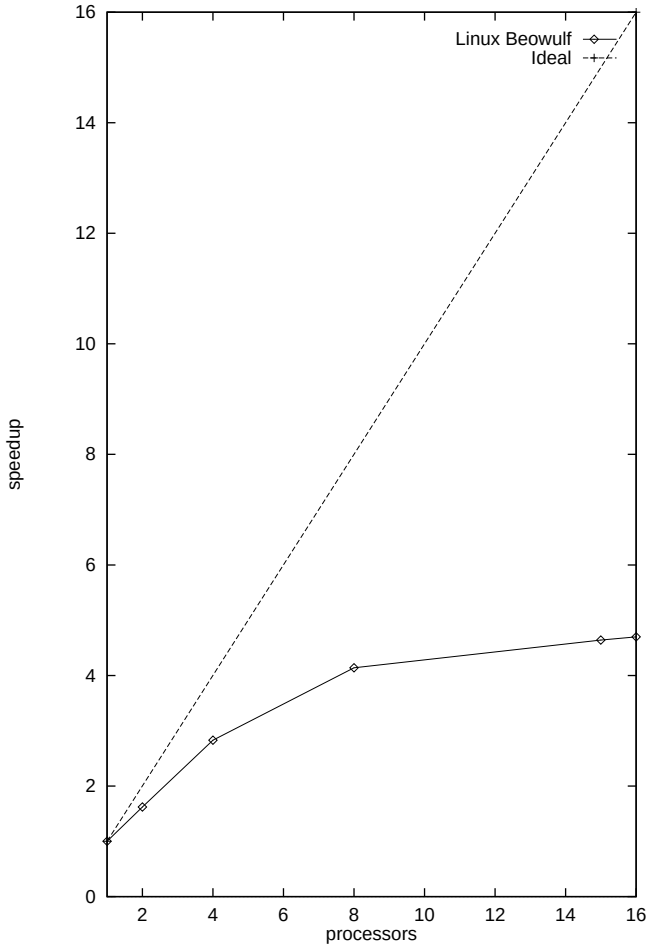

Fig. 9. Speedup for a jet problem (Intel Linux)

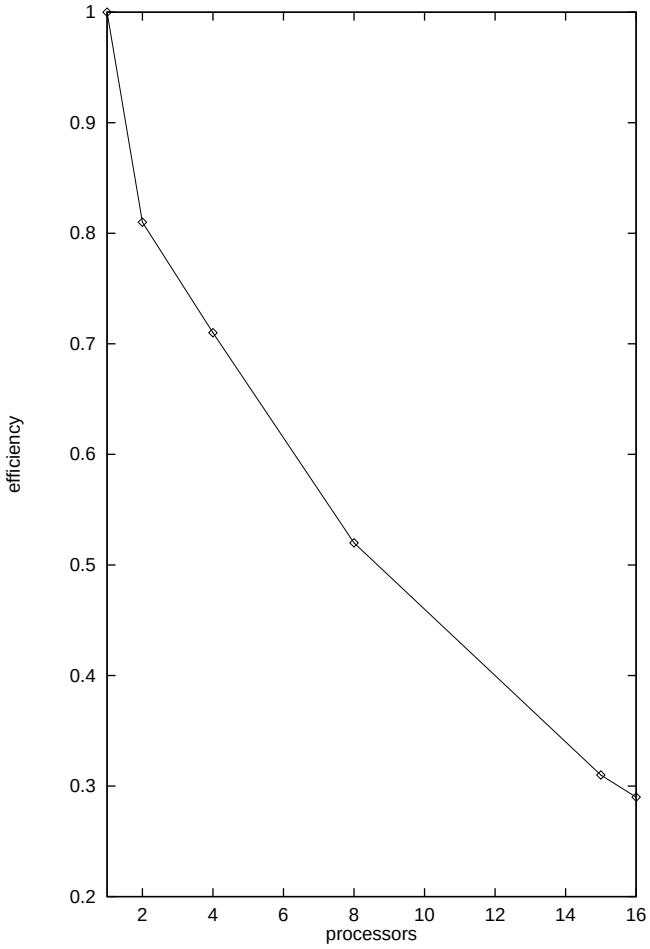

Fig. 10. Efficiency of a jet problem (Intel Linux)

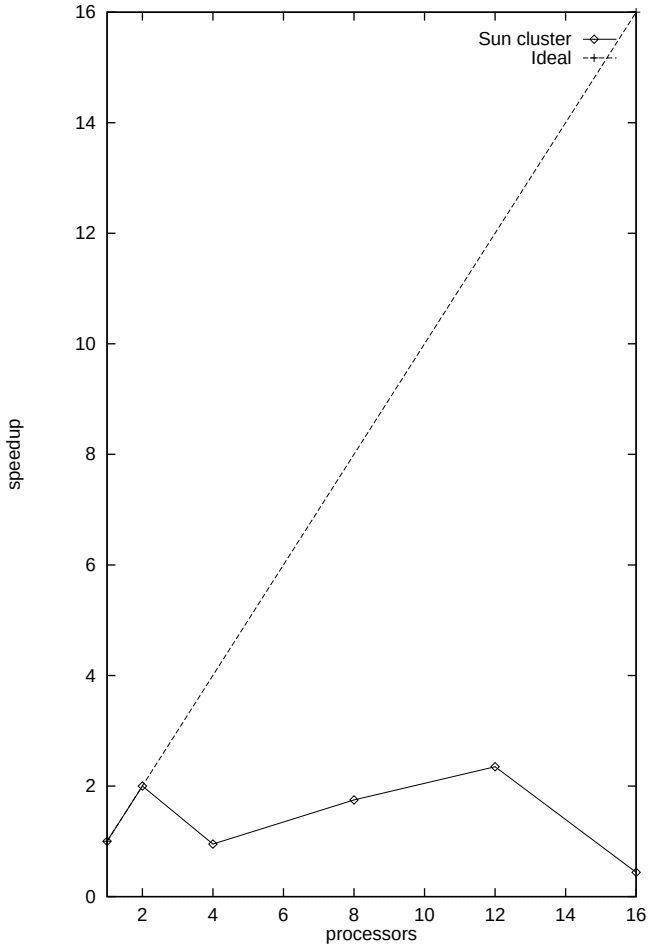

Fig. 11. Speedup of a jet problem (Sun)

Table 1. 8 Processor Jet Domain Decomposition (no periodicity)

	$128 \times 32 \times 32$ grid		$248 \times 58 \times 58$ grid	
topology	elapsed time	points/comm	elapsed time	points/comm
$4 \times 2 \times 1$	2439	7168	9456	24476
$8 \times 1 \times 1$	2009	7168	1026	23548
$2 \times 2 \times 2$	2868	9216	1115	32132

1 presents runtimes utilizing 8 processors on a Linux Beowulf cluster. The table presents runtimes for two grid sizes, each in three topologies. The results illustrate that as the problem size increases, different topologies may yield better runtimes. Some calculations can be performed to understand these trends. For the $8 \times 1 \times 1$ topology in the $128 \times 32 \times 32$ grid, each processor's domain is $16 \times 32 \times 32$ grid points. During each timestep boundary values must be communicated to each processor's neighbors. In the $8 \times 1 \times 1$ topology there are 7 boundaries that require communication. The number of grid points communicated at each time step can be found as $32 \times 32 \times 7 = 7168$ grid points. This value can be found similarly for the other topologies for the $128 \times 32 \times 32$ grid size. The $2 \times 2 \times 2$ topology has per processor domains of $64 \times 16 \times 16$ grid points in size. There are eight boundaries that communicate 64×16 grid points and four boundaries that communicate 16×16 grid points. The number of grid points communicated at each time step is $8 \times 64 \times 16 + 4 \times 16 \times 16 = 9216$. The $4 \times 2 \times 1$ topology has per processor domains of $32 \times 16 \times 32$ grid points in size. There are four boundaries that communicate 32×32 grid points and six boundaries that communicate 16×32 grid points. The number of grid points communicated at each time step is $4 \times 32 \times 32 + 6 \times 16 \times 32 = 7168$ grid points. In all of these calculations two factors of 2 could have been included for each term. One of these factors accounts for the fact that communication is bidirectional across each boundary, so twice the data accounted for above is communicated. The second factor of 2 would account for the two ghost zones in each processors domain instead of the one used above. Since these two factors of 2 would be included in each term, they can be left out for simplicity.

All of these calculations of the number of grid points communicated per time step are included in Table 1 in the column labeled "points/comm". For the $128 \times 32 \times 32$ grid, the number of grid points communicated per time step for the $4 \times 2 \times 1$ and $8 \times 1 \times 1$ topologies are the same. This is because the geometry of the per processor domains are the same, just rotated. Due to the great flexibility that ZEUS-MP offers, it is difficult to clearly determine why the timings for these two geometries are different while the grid points communicated per time step are the same. One speculation might be that there are 7 total boundaries in the $8 \times 1 \times 1$ topology meaning that there are 14 total communications per time step (recalling that the communications are bidirectional). In the $4 \times 2 \times 1$ topology, there are 10 boundaries leading to 20 total communications. Since the number of grid points per communication are the same it would seem that the $8 \times 1 \times 1$ topology should perform better, and this is the case as shown in Table 1. Following from this, the $2 \times 2 \times 2$ topology communicates more grid points per communication and needs to perform more communications per time step (12 bidirectional boundaries leads to 24 communications) and does have the longest runtime.

These previous speculations do not seem to apply in the case of the larger ($248 \times$

58×58) grid. In this case the $8 \times 1 \times 1$ topology has the fewest grid points communicated per time step and the fewest communications per time step, yet its runtime is greater than that for the $4 \times 2 \times 1$ topology. Although the total number of communications for the $8 \times 1 \times 1$ topology is fewer, they each involve more grid points than some of the communications of the $4 \times 2 \times 1$ topology. The additional number of communications in the $4 \times 2 \times 1$ topology may allow those communications to complete more in parallel because there are more processors involved that do not share boundaries. Even though the number of grid points communicated is greater, less waiting during communications might lead to the shorter runtime. Once again, the $2 \times 2 \times 2$ topology has the most grid points per communication, and the most communications per time step and consistently has the longest runtime.

8.4 Platform Effects

As a simple comparison, a run of the jet hydrodynamics simulation ran 2291 seconds on the Linux Beowulf cluster and 9297 seconds on the Sun cluster. The designs of these clusters are significantly different so no additional meaningful conclusions can be drawn from this timing comparison. The speedup obtained on both clusters (Figures 9 and 11) is similar. This indicates that there is nothing inherently different in the way the simulations are running on each cluster and that if the hardware were closer matched runtimes might be more similar than they are. As a reminder, it is believed that there are no architecture-specific optimizations for the hardware in either of these clusters.

9. REFLECTION

One of the initial goals of this project was to build ZEUS-MP for each of the test clusters. Generally, applications developed on *nix platforms are fairly portable, but ZEUS-MP was much more of a challenge to build for different platforms than ever imagined. ZEUS-MP has been implemented with cross platform compatibility in mind, but it was apparent that ZEUS-MP had never been built for the Linux, DEC, and Sun platforms that we planned to test with. There were several hurdles that made this cross platform compilation more difficult than it generally is.

9.1 System Libraries

Multiple system libraries were required to compile the package. No documentation exists with the package to indicate which libraries were required (and where they could be found), nor is there a configure script to search for the required libraries, reconfigure the makefile, or tell the user what they are missing. We found that the FFTW-2.1.2, HDF4.1R3, and MPICH-1.2.1 libraries were required for the modules of ZEUS-MP that we used. Additionally, particularly in the case of DEC, we encountered fatal errors linking against the required libraries.

9.2 System Dependencies

This was not a serious problem, since precompiler directives for SGI, Sun, and DEC (as well as others) were already defined in the ZEUS-MP package. Precompiler directives for Linux were missing.

9.3 Precompiler directives

To make the code feature rich and portable the developers utilized large numbers of precompiler directives. Unfortunately, we found that due to inconsistencies with comments in the FORTRAN code combined with using a C precompiler lead to much confusion. In some cases it was necessary to edit the FORTRAN source to allow for successful precompilation. Lines that began with 'c' (a comment in FORTRAN), especially if that line contained unbalanced single or double quotes ("routine's handling", for example) required editing.

9.4 FORTRAN compilers and syntax standards

This was perhaps the most difficult problem we faced and the only one that prevented us from compiling ZEUS-MP for the DEC cluster, and almost prevented compiling it for the Linux Beowulf cluster. Using the DEC FORTRAN compilers problems encountered were quote matches in comment lines, line lengths, code spanning multiple lines, and variable placement, among others. Careful editing of the code eventually allowed the generation of object code but the code and libraries never successfully linked to create a binary.

To compile the package for Linux Beowulf the GNU F77 front end to the GNU compiler was used. At the current time, this is the only F77 compiler freely available and it is not actively being developed and hasn't yet made it out of BETA versioning. The current version, 0.5.24, really only had problems with the quote matches in the comment lines. Some careful editing of the code and amiss a flurry of warnings, the GNU compiler did eventually build and link the code. These warnings mostly dealt with MPI routines being called with argument lists of differing length, or calls to the same MPI routine with arguments in the same position having a different type.

10. FUTURE DIRECTIONS

Through our observations, ZEUS-MP appeared to yield correct results and perform reasonably well for those architectures that were were able to build a binary. However, the package still seems to be in its infancy.

Certain combinations of input parameters caused consistent crashes, and other input parameters caused less reliable crashes, which are even more difficult to pinpoint. As a rewrite of the serial implementation of ZEUS, it is believed that additional bugs have been introduced into this code as well. Resolution of these bugs could build confidence in the package.

The large number of required libraries coupled with inconsistencies with FORTRAN compilers and code and the lack of documentation with the package make porting the existing code non-trivial. Consolidating the experiences that people have had porting this code and streamlining the building process with information from different platforms could help the adoption of this package.

The code is noted as being optimized in terms of the code on each node and also in the MPI calls that are used to communicate between nodes. Following from the limited portability, it does not appear that there are any architecture-specific optimizations for architectures other than SGI. It is very likely that our scaling results were not as ideal as those found by others because they were benchmarking

ZEUS-MP on SGI clusters.

Overall, ZEUS-MP has promise as a tool that will allow for solving large problems in a much more timely fashion than the serial implementation of ZEUS. The problems we encountered with ZEUS-MP were mainly rooted in using clusters not composed of SGI hardware. Hopefully, future releases of ZEUS-MP will bring smoother portability and optimizations for additional architectures.

REFERENCES

FIEDLER, R. 1997. Optimization and Scaling of Shared-Memory and Message-Passing Implementations of the ZEUS Hydrodynamics Algorithm. *http://www.supercomp.org/sc97/program/TECH/FIEDLER/INDEX.HTM*.

NORMAN, M. L. 1999. Introducing ZEUS-MP: A 3D, Parallel, Multiphysics Code for Astrophysical Fluid Dynamics.