Using a Developing MiniApp to Compare Platform Characteristics on Cray Systems

Bronson Messer
Scientific Computing Group
National Center for Computational Sciences, Oak Ridge National Laboratory
Oak Ridge, TN USA
Email: bronson@ornl.gov

Abstract—**The use of reduced applications that share many of the performance and implementation features of large, fully-featured code bases ("MiniApps") has gained considerable traction in recent years, especially in the context of exascale planning exercises. We have recently developed a MiniApp designed to serve as a proxy for the CHIMERA code that we have dubbed Ziz. As an initial foray, we have used the directionally-split hydro version of Ziz to quantify a handful of architectural impacts on Cray XK7 and XC30 platforms and have compared these impacts to results from a new Infiniband-based cluster at the Oak Ridge Leadership Computing Facility (OLCF). We will describe these initial results, along with some observations about generating useful MiniApps from extant applications and what these artifacts might hope to capture.**

Keywords-**MiniApps; application codes; performance;**

I. Introduction

The use of reduced applications that share many of the performance and implementation features of large, fully-featured code bases (MiniApps) has gained considerable traction in recent years, especially in the context of exascale planning exercises. The central conceit of MiniApps relies on the realization that many large scientific codes contain a small, countable number of hot spots that dominate their performance characteristics and that other parts of these large codes contain recurring tropes, where, though the ultimate aim might be different for each routine, the performance characteristics are quite similar [1]. The idea is to encapsulate these behaviors in a simpler, reduced version of the application, mimicking the important characteristics, but obviating the need for the user to be a developer/expert on the code itself. We have recently developed a MiniApp designed to serve as a proxy for the CHIMERA code that we have dubbed Ziz. CHIMERA [2], [3] is a multi-dimensional, multi-physics code designed to study core-collapse supernovae. The code is made up of three essentially independent parts: hydrodynamics, nuclear burning, and a neutrino transport solver combined within an operator-split approach. The hydrodynamics is directionally split, and the ray-by-ray transport and the thermonuclear kinetics solve occur after the radial sweep occurs, when all the necessary data for those modules is local to a processor. This combination of directionally-split hydrodynamics and operator-split local physics provides the context for the communication and computation patterns found in CHIMERA. The neutrino transport in CHIMERA is performed only in the radial direction (as this step is the most computationally intensive, and would preclude realistic runtimes if treated in full generality). The result is a subcommunicator-local sparse linear solve at each timestep. The nuclear composition in regions that are not in nuclear statistical is evolved via a completely local dense linear solve that is subcycled within each hydrodynamic timestep for every cell in the domain. Ziz will eventually be able to capture the behaviors of the directionally- split hydrodynamics (i.e. the restricted data transposes), the operator-split local physics, and the dense and sparse linear solves used in the kinetics and transport solves. As an initial foray, we have used the directionally-split hydro version of Ziz to quantify a handful of architectural impacts on Cray XK7 and XC30 platforms and have compared these impacts to results from a new Infiniband-based cluster at the Oak Ridge Leadership Computing Facility (OLCF). We describe these initial results, along with some observations about generating useful MiniApps from extant applications and what these artifacts might hope to capture.

II. Core-collapse supernovae and Chimera

The final act in a massive ($M \gtrsim 10\,M_\odot$ [solar masses]) star's existence is for the stellar core to suddenly collapse to super-nuclear densities. Remarkably, the result of core collapse and the formation of a BH or NS can be the violent expulsion of the rest of the star, through the transfer of a modest portion of the $\sim 10^{53}$ ergs (= 10^{46} J = 100 Bethe) change in core gravitational binding energy into the outer parts of the start to produce a ~ 1 Bethe explosion. Studying the CCSN mechanism requires understanding the dynamics of that energy transfer to connect pre-explosion stars with explosions and their observational consequences. Forty-plus years of study have not yet revealed a fully satisfactory understanding of the CCSN problem, but have revealed a set of physical, dimensional, and resolution requirements needed to transform our understanding of stellar explosions and their consequences.

After several million years of evolution and nuclear energy release, a massive star's core is composed of iron (and similar 'iron-peak' elements) from which no further nuclear energy can be released by fission or fusion. Outside the 'Fe'-core are shells representative of previous burning stages—

a silicon shell, oxygen shell, etc., out to a helium shell surrounded by an envelope of hydrogen. At the base of the Si-shell, nuclear 'burning' continues, growing the Fe-core below. When the mass of the Fe-core reaches the limiting Chandrasekhar mass, it starts to collapse. For slightly less massive stars (M~8–10 $M_\odot$) a similar collapse occurs, but with a core of oxygen and neon.

During the collapse, the inner core will become opaque to neutrinos and surpass the density of atomic nuclei ($\gtrsim 2.5 \times 10^{14}$ g cm^{-3}) reaching densities where individual nuclei merge together into nuclear matter. Above nuclear density, the nuclear equation of state (EoS) 'stiffens' and the core rebounds like an over-compressed spring, launching a rebound, or bounce, shock from the newly formed neutron star (a proto-NS). The rebound shock, initially enclosing ~0.5 $M_\odot$ of the ~1.5 $M_\odot$ Fe-core at a radius of ~10 km, progresses outward through the rest of the infalling core, heating and dissociateing the infalling nuclei to free nucleons and radiating a large burst of neutrinos. Thermal energy removed from the shocked material by neutrinos and nuclear dissociation halts the progress of the shock rendering it a standing accretion shock (SAS) with a radius of 100–200 km about 50 ms after it was launched. In the SAS state, the inner regions of the star continue to collapse and accrete through the shock, dissociate, and settle on the proto-NS. Heating due to accretion onto the proto-NS drives the emission of neutrinos of all three flavors (ν_e, ν_μ, ν_τ) and their anti-particles ($\bar{\nu}_e$, $\bar{\nu}_\mu$, $\bar{\nu}_\tau$). Below the shock, but above the proto-NS, the absorption of ν_e and $\bar{\nu}_e$ by the free nucleons below the shock results in a 'gain' region of net heating. In spherically symmetric (1D) simulations, fluid elements advect through the gain region before they can be heated sufficiently to reverse their direction and drive an explosion, thus 1D simulations of Fe-core collapse invariably end in the accretion of the entire star, a situation that is not matched by observations of CCSNe, though ONe-core collapse can lead to weak explosions even in spherical symmetry. It has been shown recently that multidimensional simulations can ameliorate this lack of heating and lead to explosions [4]

III. Chimera

Chimera is a multidimensional radiation hydrodynamics code designed perform precisely these kinds of multidimensional supernova simulations. The name Chimera originates in the applications's combination of these three, separate, mature codes. The primary code modules evolve the stellar fluid dynamics (MVH3), the "ray-by-ray-plus" neutrino transport (MGFLD-TRANS), and the thermonuclear kinetics (XNet). These three "heads" are augmented by a sophisticated equation of state for nuclear matter, and a self-gravity solver capable of an approximation to general-relativistic gravity. *Operator splitting* (solving physics components separately, rather than simultaneously) and *dimensional splitting* (solving equations separately for the three spatial dimensions: r, θ, ϕ) are the key themes to Chimera's computational approach to the multi-dimensional supernova problem.

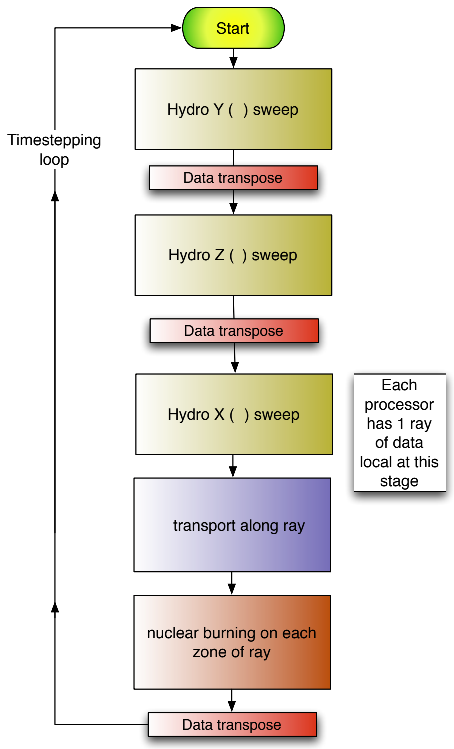


Figure 1. A schematic of Chimera program flow.

A. Fluid dynamics

Hydrodynamics are evolved using a dimensionally split, piecewise parabolic method (PPM), a version of the publicly available astrophysics PPMLR hydrodynamics code VH1. For the radial direction (r) the grid moves to track mean fluid motion and appropriately resolve features like shocks and the steep density feature at the edge of the proto-neutron star. Self-gravity is computed via multipole expansion in spherical harmonics with a global reduction to gather the coefficients needed by each processor to compute the gravitational force and potential. Spherical symmetric corrections to gravity for GR replace the non-GR (Newtonian) monopole ($\ell = 0$) term.

B. Nuclear physics

In the lower density regions that comprise the outer half of the stellar core, physical conditions require the use of a nuclear network to evolve the time-dependent abundances of nuclei. The nuclear network incorporated into Chimera is the publicly available nuclear network code XNet. XNet solves, for each non-equilibrium zone, a coupled system of non-linear ODEs (one for each nuclear species) for the time evolution of the nuclear abundances by the backward Euler method. Where the nuclear time step is smaller than the hydrodynamic step, XNet will compute multiple sub-steps automatically to prevent the short reaction timescales in a few zones from severely restricting the global simulation time step. Most Chimera simulations have used a 14-species α-network (^{4}He,^{12}C–^{60}Zn). (We have planned some complementary 3D simulations using larger networks for the XK7 'Titan' at Oak Ridge under the DOE INCITE program.) We also use passive Lagrangian tracer particles as data samplers for use in post-processing nucleosynthesis with 4000+ isotope network with XNet and other analyses. Several hundred thousand tracers are used for production simulations.

C. *Neutrino transport*

Transport of neutrinos is computed as multi-(energy)-group angular moments of the neutrino distribution function in a diffusive approximation flux-limited to prevent aphysical (superluminal) 'diffusion' of neutrinos in semi-transparent and transparent regions (multi-group flux-limited diffusion, or MGFLD). The MGFLD equations, including local couplings between all energy groups (via scattering), neutrinos and anti-neutrinos (via pair emission processes), and to the local matter, are solved by Newton-Raphson iteration independently on each radial 'ray' per the ray-by-ray approximation. The neutrino–matter interactions are a modern set that include non-isoenergetic scattering on electrons and free nucleons, emission and absorption on free nucleons and an ensemble of nuclei in NSE, and neutrino–anti-neutrino pair emission from nucleon-nucleon bremsstrahlung and e^+e^--annihilation. The MGFLD equations are coupled integro-partial differential equations. The left (or transport) side of the MGFLD equations include spatial and energy partial derivatives, while the right (or collision integral) side includes local couplings between all energy groups (via scattering), neutrinos and anti-neutrinos (via pair emission processes), and to the local electron abundance (Y_e, via emission and absorption). The resulting Jacobian has a block tri-diagonal structure. The off-diagonal blocks are themselves diagonal and contain only the discretized radial derivative terms. (Our use of the 'ray-by-ray' approximation in the transport removes the pairs of off-diagonal blocks that would be created by the omitted θ and ϕ derivatives and the inter-process communication that they would create.) The dense diagonal blocks include all of the local couplings from the collision integral. In typical production runs, there are 81 unknowns (20 energy groups each for 4 species of neutrinos plus the Y_e) in each block and ~500 blocks (one for each radial zone). The linear system from the Newton-Raphson iteration is solved by the stabilized bi-conjugate gradient method using an ADI-like preconditioner.

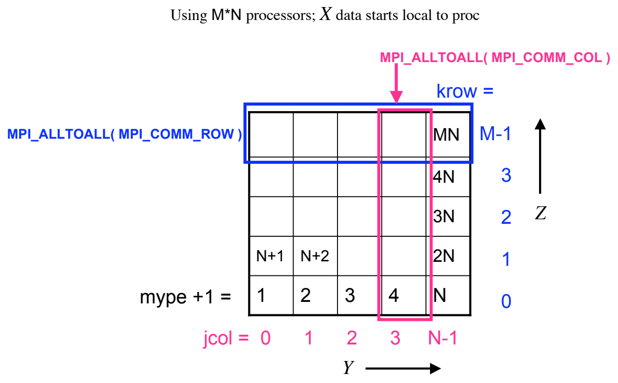
Neutrino-matter interactions (opacities) are interpolated from tables in (ρ, T, Y_e) and terms from the collision integral are computed for each Newton-Raphson iteration. The opacity data is computed during the run as needed and stored as a shared data set of the required points for each MPI-rank.

D. *Grid*

The spherical coordinates used in Chimera are natural for the CCSN problem given the centrally concentrated proto-NS and the ray-by-ray approximation. They result in similar conditions on each processor for the most expensive computational elements (transport and nuclear burning) and thus a natural load balancing of the computational work. Spherical coordinates come with a price: restricted zone sizes (and time steps) from coordinate convergence at the center and the pole. The center in Chimera (and similar codes) is treated (pseudo-)spherically, which suppresses non-radial motions in the inner core of the nearly spherical proto-NS and relieves the simulation of the time step restrictions for the non-radial zone sizes inside the core. The convergence of the grid at the poles ($\theta = 0, \pi$) is the other limiting factor (the time step restricting length for the zone closest to the pole is $\Delta\ell = R\,\Delta\phi\,\sin\theta_c$). We (and the Garching group) are implementing the pole-free, overset 'Yin-Yang' grid [5] to avoid the pole-induced time step restrictions. In the Yin-Yang grid, the zones within 45° of the pole and for one-quarter of the ϕ zones along the equator are omitted and two such grids are rotated to fit together without gaps. The zones in the Yin-Yang grid are quite similar in size, which drastically reduces the time step restrictions near the omitted pole. For a 1° simulation, the Yin-Yang grid eases the non-radial Courant limiting time step by 80-fold (the difference between $\sin 0.5°$ and $\sin 45°$).

E. *Domain decomposition with MPI*

The primary sub-domain used in Chimera is a 'ray'— a thin domain that spans the entirety of one dimension but is only one zone wide in the other dimensions. All of the dimensionally split work is done on independent rays in each of the directions (r, θ, ϕ). To pass data to and from the native radial-ray orientation of Chimera for computing along the angular dimensions, we use a system of sub-communicators. Sub-communcators are constructed by taking a group of MPI tasks that cover the full domain in r and one of the angular dimensions (θ, for example) while covering the same narrow extent in the other angular dimension (ϕ, as thin a one zone). Data within this 'slab' of zones is transposed by `MPI_AllToAll` on a sub-communicator for that 'slab' that consists of $\mathcal{O}(100)$ tasks, with a similar number of independent sub-communicators to cover the entire problem domain. After the require computations are computed in the angular dimension (θ this example) the data is transposed back to the original radial orientation by a second `MPI_AllToAll` on the same group of sub-communicators. A similarly constructed, but different, set of sub-communicators is require for the other angular dimension (ϕ) and a pair of such transposes are required for each of our $\mathcal{O}(10^6)$ time steps. All-to-all time is kept manageable



Using M*N processors; X data starts local to proc
MPI_ALLTOALL(MPI_COMM_COL)
krow =
MPI_ALLTOALL(MPI_COMM_ROW)
MN
M-1
4N
3
3N
2
Z
N+1
N+2
2N
1
mype +1 =
1
2
3
4
N
0
jcol = 0
1
2
3
N-1
Y

Figure 2. The domain decomposition and construction of transpose sub-communicators in Chimera.

(only a few percent of the total runtime) by the small size and independence of the sub-communicators.

Advection of the tracer particles requires a `MPI_AllToAllV` (the distribution of particles is not uniform and variable) within the 'slab' of the tracer position into each direction during each time step. Time step determination, conservation monitoring, and the computation of 1D average values for computing radial grid motion require global reductions (some with broadcasts of computed consequences) during each time step. The scalable MPI design for Chimera has remained viable after 10 years with only a few minor tweaks required to scale to $O(10^5)$ cores. Typically we place a single radial ray on each MPI task (~500 zones). This is the maximal strong scaling permitted in Chimera using MPI only.

F. Threading with OpenMP and overall code complexity

Two components represent the largest computational effort: the neutrino transport and the nuclear network (in the case of the larger network). Our threading efforts have concentrated on these two areas with the near-term goal of bringing more computational elements to bear on the smallest MPI-only domains to reduce the time to solution. Threading of the nuclear network in Chimera was completed in 2012 and shows good efficiency (3.6 speed-up for 4 cores/task) [3]. The recently implemented 'ADI-like' linear solver for the neutrino transport is fully threaded and shows similar speed-ups.

The complexity and size of Chimera (currently a little more than 300K LOC) makes gauging the effect of new programming models and new hardware constraints difficult. Although Chimera represents upwards of 10% of the total INCITE workload each year at the Oak Ridge Leadership Computing Facility (OLCF), the use of the code by vendors, computer scientists, and other non-astrophysicists is severely hampered by this complexity. We seek to alleviate these difficulties through the construction of a so-called MiniApp based on Chimera.

IV. MiniApps and Ziz

In general, there is a fundamental tension between the size and complexity of modern, large-scale scientific application codes and the design and fielding of new computational platforms on which these codes are run. The ultimate utility of current petascale and future exascale platforms will be judged based on the scientific productivity of their users. This means that modifying and improving the algorithms and implementations used in these large-scale codes to allow them to run efficiently on these new platforms is of primary importance. It is equally clear that a thorough understanding of the requirements and limitations of current and planned software features is absolutely essential in system design. However, such optimization work is often arduous because of the sheer size of the code bases, the complexity of the codes themselves, and, in many cases, their associated build systems. This often means that accessible benchmarks are not available at any given instant in the software and hardware development processes for vendors to optimize system design points against, for computer science and applied mathematics researchers to use as laboratories to test new algorithmic and implementation ideas, and even for application developers themselves to use as effective test mechanisms to investigate the impacts of new hardware and programming model developments. For these reasons and others, the use of reduced applications that share many of the performance and implementation features of large, fully-featured code bases (MiniApps) has gained considerable traction in recent years, especially in the context of exascale planning exercises. The central conceit of MiniApps relies on the realization that many large scientific codes contain a small, countable number of hot spots that dominate their performance characteristics and that other parts of these large codes contain recurring tropes, where, though the ultimate aim might be different for each routine, the performance characteristics are quite similar [1]. The idea is to encapsulate these behaviors in a simpler, reduced version of the application, mimicking the important characteristics, but obviating the need for the user to be a developer/expert on the code itself.

It is interesting to note that MiniApps are often described via exclusion, i.e. by enumerating the things that are not MiniApps. [6] characterize MiniApps by noting that they are not to be confused with compact applications wherein a particular implementation of physics is captured in isolation nor are they so-called skeleton applications, designed reproduce a particular pattern of inter-process communication (a goal so narrow that the computation performed between communication epochs is sometimes "faked"). [7] expands this list of what MiniApps are not to include Scalable Synthetic Compact Applications (SSCA), which were produced through the DARPA High Productivity Computing Systems (HPCS) program to evaluate the productivity of emerging HPC systems. The NNSA Exascale Applications Working Group noted [8] that despite these attempts to constrain the definition of MiniApps, their role in cooperative R&D efforts (e.g. in so-called co-design efforts, where hardware and software designers engage directly with application programmers to improve the design of both architectures and algorithms for future systems) require them to be several things all at once. They must big enough to accurately model full application behavior, but small enough to be manageable and, in some sense, parsable, by researchers ranging from applied mathematicians to compiler writers.

Related to this is the question of whether MiniApps can be effectively divorced from the underlying physics being modeled by a particular application code (see, e.g., the discussion of this point in [9]). We posit that, in general, this notion is not a useful operating assumption, based on our extensive past experience with transitioning large application codes across many orders of magnitude in scalability and performance. Indeed, we note that the realization that HPCCG (one of the original components of the Mantevo MiniApp suite [1] could only, provide a stronger tie to applications of interest, by realizing that, "the context in which the linear system is formed needed [to be](sic) strengthened" [9] suggests a stronger tie between performance and intent is often necessary to effectively understand full application performance.

An ever-expanding group of MiniApps has grown up in recent years. The Mantevo suite [1] was perhaps the first and largest set of MiniApps. Mantevo includes MiniApps designed to mimic the performance of finite-element codes (MiniFE), molecular dynamics codes (MiniMD), electrical circuit design codes (MiniXYCE), and others. It is important to note that the particular choice of MiniApps in the Mantevo suite is a direct result of the interests and expertise of the application community at Sandia National Laboratories, where the suite was developed. Other MiniApps have been produced at Los Alamos National Laboratory, e.g. CGPOP [10] and MCMini [11], and at Argonne National Laboratory (see, e.g. the MiniApps list at the CESAR website, https://cesar.mcs.anl.gov/content/software). We have started a project at OLCF in the past year designed to assemble a similar collection of codes, particular to the user community of the OLCF, that can be changed, revised, or even retired as architectures and software systems evolve.

Part of that initial suite, Ziz[1] is a CHIMERA MiniApp that currently captures the behaviors of the directionally-split hydrodynamics (i.e. the restricted data transposes). Additions to Ziz are planned to model the calculation of operator-split local physics and the dense and sparse linear solves used in the kinetics and transport solves. The resulting MiniApp can be expected to useful as a proxy for a whole class of multiphysics codes. Individual modules for nuclear burning, sparse linear system solution, and hydrodynamics (specifically, flux reconstruction at finite-volume interfaces) will be produced to allow modular CPU-GPU MiniApps to be constructed.

V. Experiments with Ziz

We have performed scaling studies for Ziz on 3 different platforms at OLCF – Titan, Eos, and Rhea.

Titan is a Cray XK7 composed of 200 cabinets. Titan is a hybrid architecture where each individual compute node uses both a conventional 16-core AMD 6274 Opteron CPU connected to 32GB of 1600 MhZ DDR3 SDRAM and an NVIDIA K20x (Kepler) GPU with 6 GB of GDDR5 memory. Titan, with 18,688 of these hybrid compute nodes, has a theoretical peak computational performance of more than 27 PFLOPS. Each of the compute nodes is interconnected with Crays high-performance, 3D-torus Gemini network.

Eos is a 744-node Cray XC30 cluster. The system has (2) external login nodes. The compute nodes are organized in blades. Each blade contains four nodes connected to a single Aries NIC. Every node has 64 GB of DDR3 SDRAM and two Intel Xeon E5-2670 CPUs with 8 physical cores each. Intels Hyper-threading (HT) technology allows each physical core to work as two logical cores so each node can functions as if it has 32 cores. Each of the two logical cores can store a program state, but they share most of their execution resources. In total, the Eos compute partition contains 11,904 traditional processor cores (23,808 logical cores with Intel Hyper-Threading enabled), and 47.6 TB of memory.

Rhea is a 196-node commodity-type Linux cluster. Each of Rhea's nodes contain two 8-core 2.0 GHz Intel Xeon E5-2650 processors with Hyper-Threading and 64GB of main memory. The nodes are connected via Mellanox 4X FDR Infiniband MT27500 network controllers.

Our initial experiments with Ziz have centered on investigating the degree to which the scaling performance of CHIMERA is mimicked by the MiniApp and how that scaling behavior changes when the interconnect is changed from Gemini to Aries to Infiniband. A comparison of CHIMERA weak scaling to Ziz "hydro-only" weak scaling on Titan is shown in Figure 3. The "hydro-only" qualification refers to

[1]A ziz is is a giant griffin-like bird in Jewish mythology, often portrayed as something somewhat akin to a Greek chimera. The naming of the MiniApp is in keeping with the CHIMERA collaboration's seemingly semi-literate wordplay and garbled understanding of Western mythologies. The name is also short and relatively difficult to mistype.

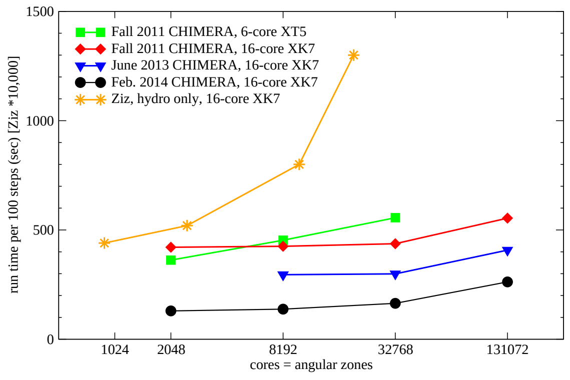


Figure 3. A comparison of hydro-only Ziz scaling and historical Chimera scaling experiments. The ordinate (i.e. time required for 100 time steps) is multiplied by 10000 for the Ziz results: The time required to perform a hydro-only update is dwarfed by the transport and burning solves in the full Chimera code.

a version of Ziz wherein only the hydrodynamics module is included, i.e. there is no active module for radiation transport or for nuclear kinetics. As expected, the weak scaling behavior of this version of Ziz becomes less efficient much more quickly than does the more physics-laden full Chimera runs. Previous profiling of Chimera has shown that the hydrodynamics calculation (including the requisite `MPI_AllToAll` communications) represents roughly 33% of the total runtime in Chimera for runs at less than O(30K) cores. For those same runs, roughly 45% of the runtime is dedicated to the transport solve and ≈20% is spent in the nuclear burning module, the balance being spread among other routines, including I/O. The transport and nuclear burning solves also significantly increase the payload sizes for the transpose `MPI_AllToAlls`, increasing the number of double-precision REALs per zone from 6 to 180 (160 for the transport and 14 for the nuclear burning). Nevertheless, it is important to understand the hydro-only scaling of the code, to better understand the turnover in the parallel efficiency that occurs above O(30K) cores.

Comparing scaling results on different machines presents a somewhat different picture, however. Shown in Figure 4 are a subset of the same Ziz runs performed on Eos and on Rhea, along with the Titan scaling. The Eos and Rhea results are restricted in number because of the size of each of the machines and the particular modulo arithmetic that must be satisfied for each dimension in a Ziz (or Chimera) run (see Section III-E above for an explanation). Perhaps unsurprisingly, Eos is (a) somewhat faster for a given number of MPI ranks and (b) weakly scales all the way out to 10,000 ranks with excellent efficiency. The runtime on Rhea is significantly longer than either of the Cray platforms,

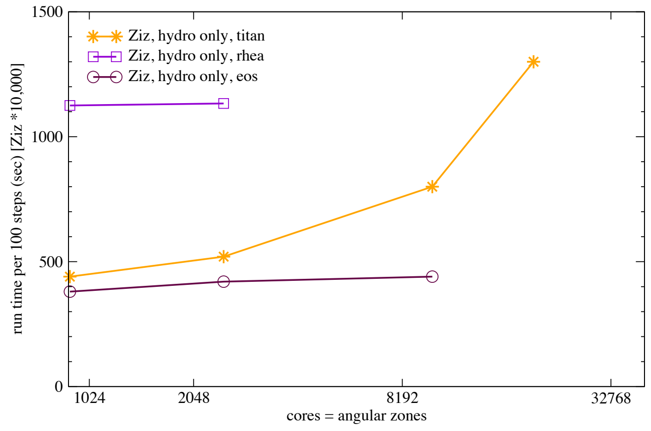


Figure 4. A comparison of hydro-only Ziz scaling on Titan, Eos, and Rhea. The ordinate (i.e. time required for 100 time steps) is multiplied by 10000 for the Ziz results, just as in Figure 3.

but the weak scaling is not quite as bad as for Titan at low rank counts. It should be noted that the Cray platform results were both obtained with code compiled with the CCE compiler suite (CCE 8.2.2 on Titan and CCE 8.1.9 on Eos), while the Rhea results came from code compiled with Intel ifort (version 13.1.3). Neither the Eos nor the Rhea results allowed HyperThreading. Additional tests on Eos with Intel ifort (version 13.1.3) produced results within 3% of the CCE results presented in Figures 3 and 4.

A. *Mimicking additional computational intensity*

In an attempt to better represent the additional computational load provided by the transport and nuclear kinetics modules of Chimera we added a fixed number of additional floating-point operations to Ziz directly after each call to `sweepx1` and `sweepx2` (the two subroutines that handle the radial sweeps in each half-timestep, including the transposes; essentially the two halves of the middle box in Figure 2). Assuming that the preponderance of the FLOPs in the transport are found in the LU decomposition of the diagonal blocks and that the kinetics solve is similarly dominated by LU decompositions for each radial zone's network, we added roughly 70 MFLOPs/zone to Ziz at this point in the program flow. The corresponding scaling for Titan and Eos is shown in Figure 5. A curious feature emerges: Although the overall weak scalability on Titan and Eos is better with the additional local work, the overall performance of Eos relative to Titan as substantially degraded. In fact, the actual wall time to solution for Eos now *exceeds* that for Titan. Though we have no established cause for this inversion, we do offer one additional observation.

Profiling this version of Ziz with CrayPat allowed us to determine the breakdown of time dedicated to computation and communication versus the hydro-only version. This comparison is shown in Table I. `fakeflops` is the function containing the additional computational work. The stark

Table I

Ziz CrayPat Profiling - Top functions as % of time

platform	hydro only	added FLOPs
Titan	`MPI_AllToAll` = 38%, `parabola` = 20%	`MPI_AllToAll` = 45%, `fakeflops` = 45%, `parabola` < 1%
Eos	`MPI_AllToAll` = 30%, `parabola` = 14%	`MPI_AllToAll` = 78%, `fakeflops` = 19%, `parabola` < 1%

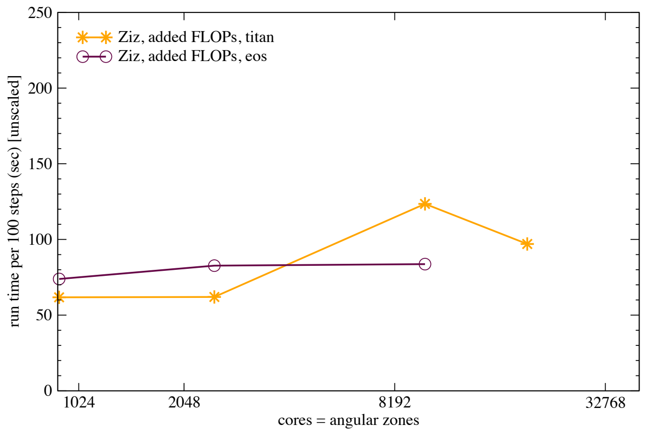


Figure 5. A comparison of the artificially FLOP-intensive version of Ziz scaling on Titan and Eos. Note that, unlike earlier figures, the ordinate has not been scaled here.

difference in the profiles of the FLOP-rich versions on Titan and Eos suggests that load imbalance on EOS might be the culprit in the hunt for an explanation of the timing inversion. The final answer awaits further investigation.

B. *Initial work on an OpenACC Ziz version*

There is a beta OpenACC version of Ziz under active development. This port is based on a version of MVH3 that was used as the "code lab" for an early 2012 OLCF GPU programming tutorial. Aside from a handful of loop re-orderings that proved necessary for the port, little additional coding is required. Because Ziz relies heavily on the use of global variables declared and managed in Fortran modules (as does Chimera), care must be taken to properly scope `private` and `copyin` variables for `acc parallel` loops. Aside from these modest code changes, the only additional code present in the OpenACC version of Ziz is the addition of `acc parallel loop` directives around the single loops containing calls to the main PPM function (`ppmlr`) in the each of directional sweep routines (`sweepx1, sweepx2, sweepy`, and `sweepz`). `ppmlr` contains the most computationally intensive hydro routine – `parabola` – which is already highly vectorizable. The result is a 94% increase in the overall per-rank performance on Titan:

```
OpenAcc Ziz
speed = 887.4 kz/s/pe
Ziz
speed = 457.1 kz/s/pe.
```

The next step for the OpenACC port of Ziz will be to add accelerated versions of the LU decompositions performed in the transport and burning modules. Because these changes will, for the most part, be confined to formation of the individual Jacobians and the use of accelerated libraries, the expected code changes for these additions is expected to be just as modest as those performed for the hydro module.

The architectural differences between the Cray XK7 and XC30 platforms include both a different processor and a different network technology. These two, intertwined facets of each platform can impact the performance of even a reduced application like Ziz in unexpected ways. Layering complexity in the Ziz MiniApp will allow each of the individual pieces of multiphysics modeled by the code to be measured and analyzed separately. In addition, the pairwise and higher-order interactions between the various modules can also be delineated in this manner.

Acknowledgment

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This research was sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy.

References

[1] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, 2009.

[2] O. E. B. Messer, S. W. Bruenn, J. M. Blondin, W. R. Hix, and A. Mezzacappa, "Multidimensional, multiphysics simulations of core-collapse supernovae," *Journal of Physics Conference Series*, vol. 125, p. 012010, 2008.

[3] O. E. B. Messer, J. A. Harris, S. T. Parete-Koon, and M. A. Chertkow, "Multicore and Accelerator Development for a Leadership-Class Stellar Astrophysics Code," *Lecture Notes in Computer Science*, vol. 7782, p. 92, 2013.

[4] S. W. Bruenn, A. Mezzacappa, W. R. Hix, E. J. Lentz, O. E. B. Messer, E. J. Lingerfelt, J. M. Blondin, E. Endeve, P. Marronetti, and K. N. Yakunin, "Axisymmetric Ab Initio Core-Collapse Supernova Simulations of 12-25 $M_\odot$ Stars," *The Astrophysical Journal*, vol. 767, p. L6, Apr. 2013.

[5] A. Kageyama and T. Sato, “Yin-yang grid: An overset grid in spherical geometry,” *Geochemistry, Geophysics, Geosystems*, vol. 5, no. 9, 2004.

[6] R. F. Barrett, M. A. Heroux, P. T. Lin, C. T. Vaughan, and A. B. Williams, “Poster: Mini-applications: Vehicles for co-design,” in *Proceedings of the 2011 Companion on High Performance Computing Networking, Storage and Analysis Companion*, ser. SC ’11 Companion. New York, NY, USA: ACM, 2011, pp. 1–2.

[7] S. Dosanjh, R. Barrett, M. Heroux, and A. Rodrigues, “Achieving exascale computing through hardware/software co-design,” in *Proceedings of the 18th European MPI Users’ Group Conference on Recent Advances in the Message Passing Interface*, ser. EuroMPI’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 5–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2042476.2042478

[8] R. Springmeyer, C. Still, M. Schulz, J. Ahrens, S. Hemmert, R. Minnich, P. McCormick, L. Ward, and D. Knoll, “From Petascale to Exascale: Eight Focus Areas of R&D Challenges for HPC Simulation Environments,” *Lawrence Livermore National Laboratory, Tech. Rep. LLNL-TR-474731*, 2011.

[9] R. F. Barrett, P. S. Crozier, D. W. Doerfler, S. D. Hammond, M. A. Heroux, P. T. Lin, H. K. Thornquist, T. G. Trucano, and C. T. Vaughan, “Summary of Work for ASC L2 Milestone 4465: Characterize the Role of the Mini-Application in Predicting Key Performance Characteristics of Real Applications,” *Sandia National Laboratories, Tech. Rep. SAND2012-4667*, 2012.

[10] A. Stone, J. M. Dennis, M. M. Strout, A. Stone, J. M. Dennis, and M. M. Strout, “The CGPOP Miniapp, Version 1.0,” *Colorado State University, Technical Report CS-11-103*, 2011.

[11] R. Marcus, “MCMini: Monte Carlo on GPGPU,” *Los Alamos National Laboratory, Tech. Rep. LA-UR-12-23206*, 2012.