Title:	Ristra Project FY23 L2 Milestone Report, rev. 1: MRT #8541: Multiphysics Scaling on EAS-3
Author(s):	Malone, Christopher M.
Intended for:	Report
Issued:	2024-06-17

Ristra Project FY23 L2 Milestone Report

rev. 1

MRT # 8541: Multiphysics Scaling on EAS-3

Chris Malone, XCP-1
on behalf of the Ristra team:

Paul Barclay, Richard Berger, Dennis Bowen, William Dai, Adrian Diaz, Travis Drayna, Philipp Edelmann, Rao Garimella, Jonathan Pietarila Graham, David Gunter, Angela Herring, Davis Herring, Sumathi Lakshmiranganatha, Konstantin Lipnikov, Ollie Lo, Julien Loiseau, Louie Long, Mack Kenamond, Pat McCormick, Scott Pakin, Hoby Rakotoarivelo, Navamita Ray, Andrew Reisner, Dave Rich, Kyle Roarty, Misha Shashkov, Galen Shipman, Henry Stam, Karen Tsai, Nathan Vaughn-Kukura, Jiajia Waters, Duan Zhang.

LA-UR-24-XXXXX
June 10, 2024

Los Alamos
NATIONAL LABORATORY

Prepared for:	NNSA	
	ASC Program Office	
	Emily Simpson	Emily.Simpson@nnsa.doe.gov
	Si Hammond (CSSE)	Simmon.Hammond@nnsa.doe.gov
	James Peltz (IC)	James.Peltz@nnsa.doe.gov
	LANL	
	ASC Program Office	
	Jason Pruet	jpruet@lanl.gov
	Aimee Hungerford	aimee@lanl.gov
	Tabitha Lopez	tabitha@lanl.gov
Prepared by:	Chris Malone	
	XCP-1	

Contents

Figures

Tables

Executive Summary

The findings of this report were used to close out the ATDM milestone MRT# 8541, which was designed to demonstrate readiness of ATDM multiphysics codes for mission-relevant work on ATS-4, El Capitan. To this end, the closure criteria were to run a 3D shaped charge problem at scale up to 50% of the El Capitan early-access system, RZVernal (AMD Trento CPUs and AMD MI-250X GPUs), demonstrate scalability, and document challenges with the software stack and environment. LANL's approach to this milestone was to test our modular software capability by developing an entirely new code, `Moya`, built upon our `FleCSI` framework. The physics capability and the GPU infrastructure needed for the shaped charge problem on GPUs was added to `Moya`, and the required calculations were performed at scale. `Moya` showed good scaling without any fine-tuning of GPU kernels; there is still significant room for performance enhancements, especially for the `Legion` backend. Tied up in this L2 milestone was a closeout of KPP-3s for the ECP ST Projects at LANL; this material will be covered in a separate document.

1 Introduction

This milestone centered around the readiness of ATDM multiphysics applications to utilize the upcoming El Capitan exascale supercomputer for mission-relevant work. El Capitan is one of the first large-scale HPC machines in the complex that leverages both AMD CPUs and AMD GPUs; previous large GPU systems have used NVIDIA GPUs. To demonstrate such readiness, each lab in the Tri-Lab complex was asked to run a representative problem at scale on early-access hardware for El Capitan, and to report their findings on both scalability and robustness of the hardware and software stacks.

For LANL's demonstration problem, we chose a 3D shaped charge calculation to run on the unclassified RZVernal (or EAS-3) system. RZVernal is a precursor to El Capitan, and includes AMD Trento CPUs coupled with AMD MI-250X GPUs. At the time of the milestone, the Confluence-published size of the machine included 32 compute nodes[1], such that we needed to scale out to 16 compute nodes.

The Ristra project houses LANL's multiphysics ATDM effort. Throughout the duration of ATDM, Ristra has taken a high-risk/high-reward approach by investigating task-based parallelism and software abstractions to give agility to software development. One goal of this effort was to explore doing software development differently, such that we create an environment whereby we can rapidly stand up new codes to answer new questions on new hardware. For this milestone, we decided to test aspects of this environment.

LANL's post-ATDM strategy involves the Ristra project developing production multiphysics applications that leverage and augment the current production code capabilities. An end goal is to eventually replace our current capabilities with these more modern production codes; if the ASCI history is any indication, this process will take at least a decade.

To combine this milestone effort with our post-ATDM strategy, the Ristra project decided to develop a new code, `Moya`, whose intention is to first target the work for the milestone and then to become an unclassified production multiphysics application focusing on low energy-density physics. `Moya` is built upon the `FleCSI` framework that has been under development at LANL since the beginning of ATDM. `Moya`'s first physics capability was committed on Dec. 23, 2022 and our milestone was demonstrated in early May 2023. The rapidity with which our physics capability was implemented, ported to GPUs, and then ported to RZVernal is a demonstration of our goal for rapid new software development on new hardware.

2 Methods

The Ristra project's trajectory since the beginning of the ATDM effort was to provide flexibility in

- performance portability to new hardware
- programming models optimally suited to the hardware and/or application
- discretization techniques.

[1]The system apparently had 36 compute nodes, but our results of 50% scaling are based on the published size.

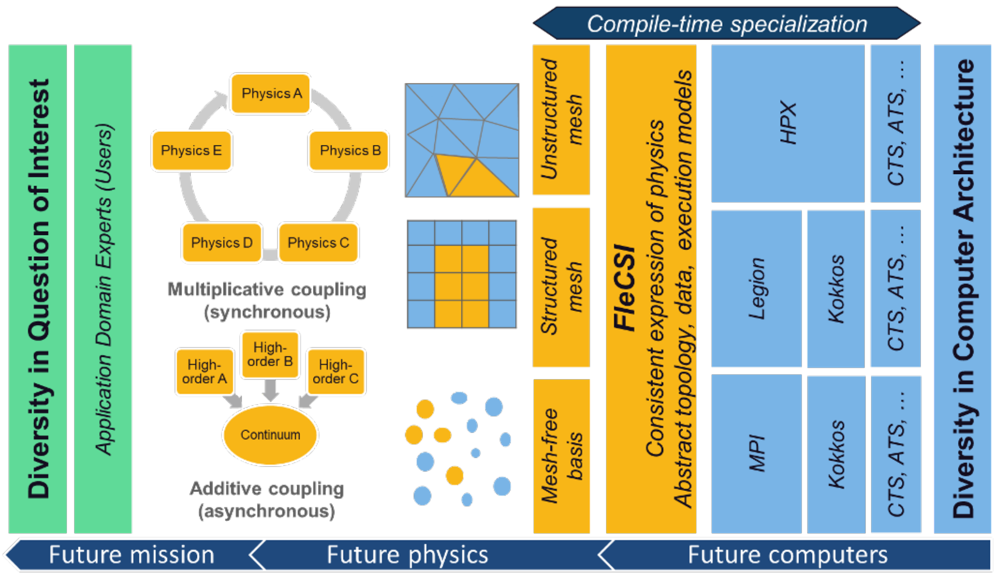
Compile-time specialization
Diversity in Question of Interest
Application Domain Experts (Users)
Physics A
Physics B
Physics C
Physics D
Physics E
Multiplicative coupling (synchronous)
High-order A
High-order B
High-order C
Continuum
Additive coupling (asynchronous)
Unstructured mesh
Structured mesh
Mesh-free basis
FleCSI
Consistent expression of physics
Abstract topology, data, execution models
HPX
Legion
Kokkos
MPI
Kokkos
CTS, ATS, ...
CTS, ATS, ...
CTS, ATS, ...
Diversity in Computer Architecture
Future mission
Future physics
Future computers

Figure 2-1: Strategy of the Ristra project, as outlined in early Ristra presentations. `FleCSI` sits at the heart of this strategy, and enables abstractions of data, programming models, and hardware.

Figure 2-1 demonstrates this strategy, which is enabled through abstraction layers. At the heart of the strategy is the `FleCSI` framework[2] - a layer designed to support multiphysics application development by abstracting task-based programming models and discretization techniques. Much of `FleCSI`'s approach to task-based parallelism is modeled after the Legion[3] programming system, which allows for describing blocks of data in memory and the tasks that need to operate on the data in such a way that the graph of tasks and their dependencies can be optimized for a given set of hardware. `FleCSI` goes further than this by providing abstract discretization concepts, which are further specialized via a *specialization layer*. As an example, a two-dimensional mesh can abstractly be thought of as a collection of two-dimensional objects (say, cells) that are connected together either through one dimensional (edges or faces) or zero dimensional objects (nodes). Each of those n-dimensional objects can hold various fields of data, including their connectivity information, which could be tailored for, say, a structured mesh. Similar concepts extend to arbitrary collections of unstructured entities or particle swarms.

For the purposes of this milestone, it is worth mentioning that Ristra, `FleCSI`, and the Specializations Project are each their own Project within ASC at LANL. They also fall under different subprogram elements.

2.1 Moya

To demonstrate the efficacy of the model picture in Figure 2-1, LANL decided that in addition to supporting such a framework, the Ristra project should develop a new production multiphysics code targeting the low energy-density application space. Our unclassified code for this task is `Moya`,

[2] https://flecsi.org
[3] https://legion.stanford.edu/

a finite-volume, arbitrary mesh code. The specialization supporting the mesh infrastructure is tentatively named the `Burton`[4] mesh. The `Burton` specialization layer was co-designed with the `FleCSI` and specialization projects, in particular adding efficient support for the subcell entities needed for the hydrodynamics methodology described below.

2.1.1 Hydro

Figure 2-2 shows a two-dimensional example of the subcell entities needed for `Moya`'s staggered-grid hydrodynamics (SGH). The method implemented is very similar to the SGH method within the Lagrangian Applications Project's `FLAG` code, where the major current differences are in the time integration. `Moya` currently uses a standard explicit 2nd order Runge-Kutta method. Adopting a standard time integration scheme (as opposed to `FLAG`'s custom predictor-corrector scheme) facilitates future exploration of higher-order or IMEX methods. The artificial viscosity model implemented is of a Von Neumann-Richtmyer form plus a linear term.

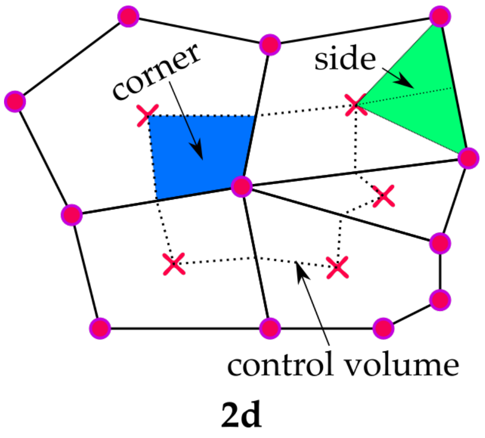

Figure 2-2: Example of an unstructured, arbitrary polytopal mesh used within `Moya`. This mesh has five cells (centers at red X) and 12 nodes (circles). Corners are used to construct the control volume for the nodes; sides are used to track normals of the control volume, and face-centered quantities. These subcell concepts extend to three dimensions.

The basic hydrodynamics method is pure Langrangian, meaning the mesh nodes move with the flow. For problems with strong shear or vortical flow, the mesh configuration can yield zones with bad aspect ratio of edges/faces, or transform cells to be non-convex or even inside out. A non-physical mechanism used to counter this is adding mesh stiffeners which resist the motion of nodes based on some cell properties; we implemented a temporary quadrilaterial subzoning approach. Mesh stiffeners can only work so much before they corrupt the solution physics for the sake of robustness of the calculation. This limits how long one can simulate in a pure Lagrangian fashion, and leads to the use of Arbitrary Lagrangian Eulerian (ALE) methods. In these methods, a step is taken to *relax* the mesh from a poor configuration to a better configuration by moving a subset of the nodes of the mesh. There are many metrics for what constitutes "poor" or "better" in this context, with corresponding methods for how one should move the nodes to achieve better metric values. As part of the milestone effort we implemented both a Laplacian smoother relaxer and a feasible set relaxer in 3D atop `FleCSI`; this was done in conjunction with the PUMA project.

Once the relaxation step has moved nodes of the mesh, the field data on the mesh needs to be made consistent with the new zone volumes. This process is called *remap*. We worked to integrate the remap capability from the PUMA project's `Portage` code, which allows for both swept-face and exact intersection-based remap.

All of the steps above need to allow for multiple materials - each with their own physics models. When dealing with multiple, immiscible materials within a zone, one needs a mechanism to keep track of the boundary between the materials. This process is called *interface reconstruction*. We

[4]After Don Burton, and the `FLAG` legacy.

integrated the PUMA project's `Tangram` capability to leverage a Volume of Fluid technique to interface reconstruction. In the future, we plan to leverage the Moment of Fluid capability of `Tangram`.

The shaped charge problem of this milestone is one involving high shear at the HE/metal boundary. As such, to run to completion and see jet formation, one *needs* to leverage ALE. Although we integrated the capabilities for remap and interface reconstruction as described above, we had issues getting the multimaterial remap working properly (i.e. conservatively) within `Moya` by the time of writing this report. This limits how long into the shaped problem we can simulate before the mesh becomes untenable, as shown below. The Ristra and PUMA teams have been working to get this fixed, as it is the planned path for ALE within `Moya`. The capability is working for simplified problems like a multimaterial Sod shock tube, but issues remain for the shaped charge that are being debugged. One capability missing for the shaped charge is the remap of high explosive light-times; those are not conserved quantities, which are the typical things one remaps.

2.1.2 Physics Models for the Shaped Charge

Physics models were prioritized to those required to run the simplified shaped charge problem. At a minimum, one needs

- equations of state (EOS) for gases, solids, and high explosives
- a strength model for the solids
- a model for igniting and burning the high explosive

to model the shaped charge. Due to `Moya`'s being a new code and the focus on RZVernal, we implemented rather low fidelity physics models. For the equations of state for gases, solids, and high explosives we used gamma law, Gruneisen, and Jones-Wilkins-Lee (JWL) models, respectively. The strength model we implemented was that of Steinberg-Guinan with an optional linear hardening term; we also added a Lindemann melt model. The high explosives were point-detonated and used a programmed burn model with direct lighting based on light-of-sight from the detonator location. It is worth noting that several of these models - although they are simple - are used in daily production calculations of mission-relevant work in existing codes.

2.1.3 Implementation Details and Porting to GPUs

All the capabilities mentioned above - aside from the remap capabilities[5] - were implemented within `Moya` on top of `FleCSI` and the `Burton` specialization layer. The specialization provides a mesh class, `mesh_t`, with a set of `index_spaces` over which one can iterate and attach `FleCSI field` objects. `index_spaces` exist for things like cells, corners, sides, etc. An example of defining the cell-centered density looks like

```
inline const field<double>::definition<mesh_t,
↪  mesh_t::index_space::cells> c_dens;
```

[5] `Portage` and `Tangram` are currently agnostic to the underlying data format by *wrapping* them in a format that they understand. This stack is currently not ported to GPUs, but that effort is in the plans for next FY.

When one wants to use a field within a *task* one has to specify permissions regarding the reading or writing to owned, shared, or ghost data. The permissions information allows `FleCSI` - with the `Legion` backend - to determine dependencies between tasks and whether or not they could be reordered. For both the MPI and `Legion` backends, the permission information can be used to determine if communication needs to happen such that one is reading from up-to-date information from peer ranks[6].

As a concrete example, consider the case of initializing the volume (`cv_vol`) and mass (`cv_mass`) of corner subcell entities from the cell density (`c_dens`) and the corresponding side volumes (`s_vol`).[7] This is a task (a regular function) that performs this operation

```
using wo_field=field<double>::accessor<wo,wo,na>;
using ro_field=field<double>::accessor<ro,ro,ro>;
void corner_state(mesh_t::accessor<ro,ro,ro> mesh,
                  ro_field s_vol, ro_field c_dens,
                  wo_field cv_vol, wo_field cv_mass) {
  for (auto cv : mesh.corners<mesh_t::owned>()) {
    cv_vol[cv] = 0.0;
    cv_mass[cv] = 0.0;
  }
  // use half-volume of each side to populate corners
  // Assumes CONSTANT density field in cell
  for (auto s : mesh.sides<mesh_t::owned>()) {
    auto c = mesh.cells(s)[0];  // cell for this side
    double svol = s_vol[s];
    for (auto cv : mesh.corners(s)) {
      cv_vol[cv] += 0.5*svol;
      cv_mass[cv] += 0.5*svol*c_dens[c];
    }
  }
}  // corner state
```

`accessors` are the mechanism by which we access both the mesh information and fields, and describe the permissions such as `ro`, `wo`, or `na` for read-only, write-only, or no access, respectively. The ordering of the permissions describes those for owned, shared, and ghost data. “Shared” data in this context are things that a particular rank (color) *owns* but that some other rank (color) needs to read as ghost information; “owned” data is exclusive to the rank (color) that owns it. The `FleCSI` and specialization-provided iterators like `mesh.corners<mesh_t::owned>` allow iterating over an index space - or a subset of only the “owned” data in this case - where field data is stored. Connectivity information is built up by the specialization layer, and exposed to the developer in methods like `mesh.corners(s)` that return all the corners attached to a

[6]“Colors” in the parlance of task-based parallelism.

[7]Every side intersects exactly two corners equally, thus a side’s volume is equally partitioned to each corner. In three dimensions, more than two sides intersecta given corner.

specific side. FleCSI provides abstractions for all these things, and the specialization makes them concrete for a particular application. Calling the above task looks like

```
flecsi::execute<corner_state>(mesh, s_vol(mesh), c_dens(mesh),
 ↪  cv_vol(mesh), cv_mass(mesh));
```

At the call site, one does not need to specify permissions. This fits nicely with the fact that the developer *implementing* a task is the one that should know more about the data usage patterns for the algorithms within the task. The caller only needs to specify on which specialization instance (a topology in FleCSI parlance) the fields are registered; one could have different representations on a mesh as in this case, but also, say, a collection of particles.

The above syntax works the same whether one is using the MPI or Legion backend of FleCSI. There are a couple of changes if one wants to leverage on-node parallelism such as through OpenMP or GPUs. First and foremost, the data types used within fields need to be GPU-friendly. The example above uses doubles for the field type, but for more complicated things like physics vectors or tensors, one needs more care. Indeed, our initial implementation of these types for CPU testing leveraged the C++ standard template library and containers like std::vector<double>, which are not universally supported on GPU hardware. To that end, we implemented some of our own classes that wrap fixed sized arrays such as

```
using Real = double;
template<class T, std::size_t D>
class Vec {
public:
  using value_type = T;
  using size_type = std::size_t;
  //  ...
protected:
  value_type data_[D];
};
using Vec3R = Vec<Real,3>;
```

which mimic a subset of capabilities from std::array. As with any GPU programming, any methods that are going to be called from the GPU device must be annotated properly for the compiler to generate correct code. FleCSI provides a FLECSI_INLINE_TARGET macro that expands to the appropriate annotations based on whether or not we are compiling for the CPU or GPU.

In terms of the actual changes to the application code and tasks, there are additional changes. First, one needs to indicate that the task is to be executed on an accelerator. Using the example above, one needs to add a template parameter to the flecsi::execute function

```
flecsi::execute<corner_state, default_accelerator>(mesh,
 ↪  s_vol(mesh), c_dens(mesh), cv_vol(mesh), cv_mass(mesh));
```

The value of default_accelerator indicates whether one is targeting a CPU, threading via OpenMP, or a GPU and is set by compile-time options; without this extra template parameter, flecsi :: execute defaults to the CPU. Similarly, one must change the contents of the the task function itself to indicate GPU code.

In Ristra, we are leveraging Kokkos for on-node parallelism. Within a task that we want to launch with the default_accelerator, we can use Kokkos commands to launch kernels on the GPU (or CPU if we are leveraging the OpenMP capability). FleCSI provides a forall macro for conveniently getting onto the GPU. The changed code looks like

```
using wo_field=field<double>::accessor<wo,wo,na>;
using ro_field=field<double>::accessor<ro,ro,ro>;
void corner_state(mesh_t::accessor<ro,ro,ro> mesh,
                  ro_field s_vol, ro_field c_dens,
                  wo_field cv_vol, wo_field cv_mass) {
  forall(cv, mesh.corners<mesh_t::owned>(), "init"){
    cv_vol[cv] = 0.0;
    cv_mass[cv] = 0.0;
  };
  // use half-volume of each side to populate corners
  // Assumes CONSTANT density field in cell
  forall(s, mesh.sides<mesh_t::owned>(), "calc"){
    auto c = mesh.cells(s)[0];  // cell for this side
    double svol = s_vol[s];
    for (auto cv : mesh.corners(s)) {
      cv_vol[cv] += 0.5*svol;
      cv_mass[cv] += 0.5*svol*c_dens[c];
    }
  };
}  // corner state
```

In the case of using Kokkos, the forall expands to a Kokkos :: parallel_for operation.[8] The default configuration expands to a Kokkos :: RangePolicy over the index space specfiied as the second argument to the forall macro. One can do further optimizations by chosing more intelligent Policies or by writing explicit Kokkos code. For this milestone, we used the default configuration as shown above without attempting any optimizations.

2.1.4 Building and Running on RZVernal

Moya is built using CMake. Dependencies are managed with Spack and the build environment for Moya uses Spack Environments. FleCSI offers some of the most challenging parts of Moya's build because of Legion and its dependencies. In particular, GASNet needed a custom, manual build for the Slingshot 11 network on RZVernal. FleCSI itself provides its own, modified Spack recipe for Legion as it relies on a custom branch of Legion for compatibility.[9] As with all

[8]The same syntax can be used without Kokkos for non-threaded, CPU-only code (e.g. in the case of an MPI-only calculation) where the forall expands to a normal loop as in the first instance of corner_state shown above.

[9]These are the so-called control-replication branches. The Legion team is working to get these capabilities merged into the mainline branch as this is required for sustainable workflows in the future.

things Spack, succcess of this process depends on compatible compilers and Spack hash. The combinatorial search space for a successful combination of builds is quite large. FleCSI requires C++-17 features, which helps limit the space of available options. On RZVernal, we ended up using hipcc from the amd/5.4.3 module as our compiler.

Getting the environment up and running on RZVernal was complicated by the fact that some of our personnel that helped with the Spack environment and debugging of issues related to Legion were foreign nationals who were not granted access to RZVernal. Instead, they had access to Tioga, the similar hardware system on LLNL's CZ network. During our rollout, Tioga and RZVernal weren't always *exactly* the same in terms of software stack. This caused some issues when our FNs work on Tioga was then brought to RZVernal to assist in actually running the milestone problem.

In terms of getting working runs, we also suffered some issues. In particular, Legion worked with Kokkos, Legion worked with HIP directly, but there were challenges getting Legion to work with Kokkos on top of HIP. We were in close collaboration with the Legion team at Stanford/SLAC to correct these issues. The first success was just a couple of weeks before our Milestone presentation deadline.

2.2 Verification and Comparisons to FLAG

To ensure that our implementations were working as expected, we ran several test problems both of the canonical type (e.g. Sod shock tube or Sedov blast wave) and of some others to compare with the FLAG code as requested by the milestone committee. Figure 2-3 shows an example of the former problem type with an octant-variant of the Sod shock tube problem. In this problem, the high pressure region representing the post-shock conditions is initialized in a single corner of the rectilinear mesh. This causes lots of shear flow along diagonal directions. This problem was used to help test our mesh stiffeners before we attempted to perform ALE calculations.

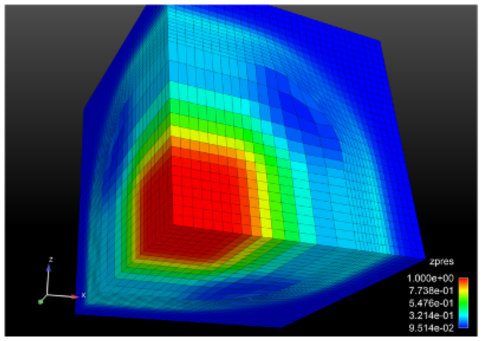

Figure 2-3: Moya calculation of an octant Sod problem with mesh stiffeners.

Figure 2-4 shows two examples of comparisons made to FLAG. The top two plots show a toy problem with three detonation points igniting and sending shocks out that interact with one another. The quantity plotted is the specific internal energy with the left result Moya and the right FLAG. The eyeball norm has these looking very similar for our simple line-of-sight burn model. The bottom plot shows calculations of an aluminum flyer plate being struck. There are four curves of the surface velocity on this plot with Moya results being the red curves and FLAG results in dashed black. The problem was run with two different strength models: Steinberg-Guinan and a linear harding model. For each model, the FLAG and Moya results lie on top of one another.

Since the milestone presentation, we have been working on our ALE capability as mentioned above. For this, we have been running several test problems with strong shear flow and multiple materials to test the ALE treatment. To this end, we have been running multimaterial variants of Sod and the triple-point problem. Results are still pending as we debug issues.

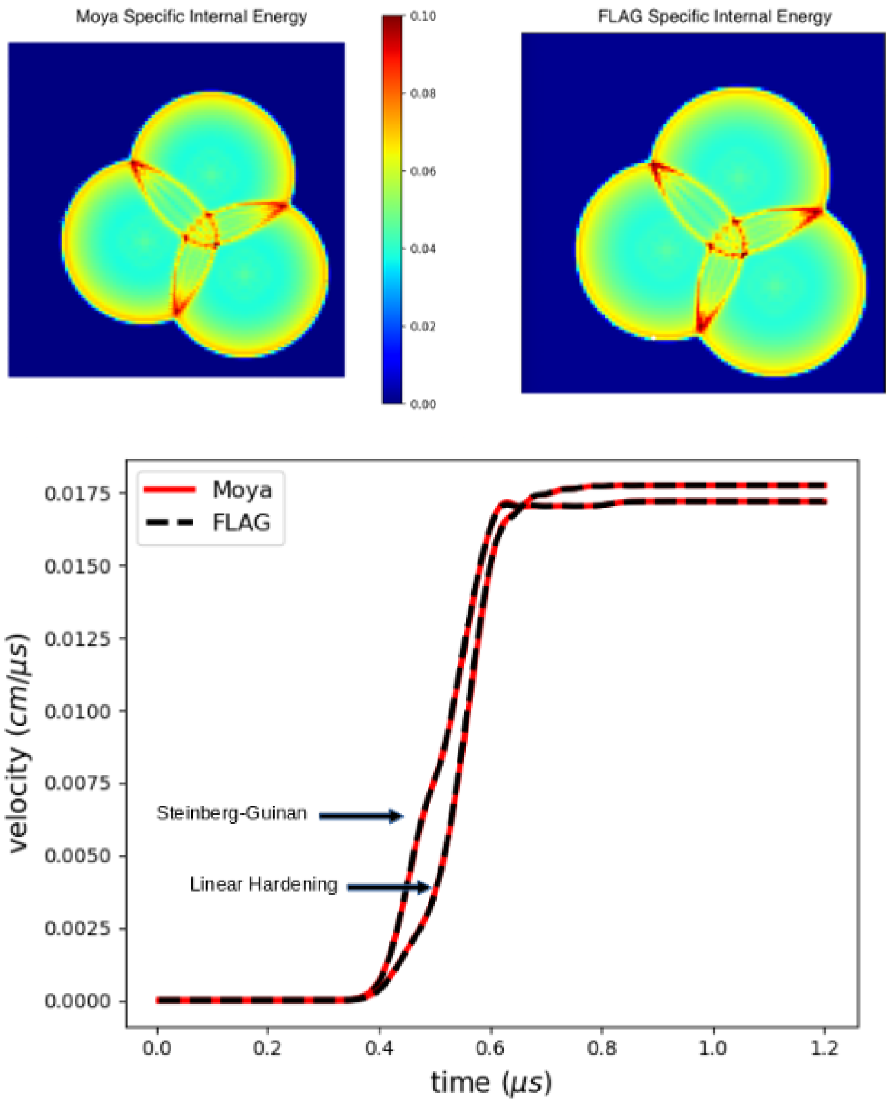

Figure 2-4: Two examples of test problems compared to FLAG. Top: a test of our HE burn machinery by igniting three detonation points. Bottom: an aluminum flyer plate impact calculation with two different model choices in both codes.

2.3 Shaped Charge Mesh

The meshes used for our shaped charge calculations were generated with LANL's in-development CrossLink tool. CrossLink is intended to give users true three-dimensional meshing capability as opposed to the current workflow of generating a two-dimensional mesh in ingen and then spinning it about an axis with charybdis. CrossLink uses a target matrix optimization technique to optimize and smooth meshes based on their topology. Figure 2-5 shows a three-dimensional mesh and a slice through that mesh for our shaped charge problem.

For the milestone, we needed to perform a scaling study. Ideally we would have liked to go up to as large of a mesh as we could fit in memory. However, due to the fact that CrossLink is still

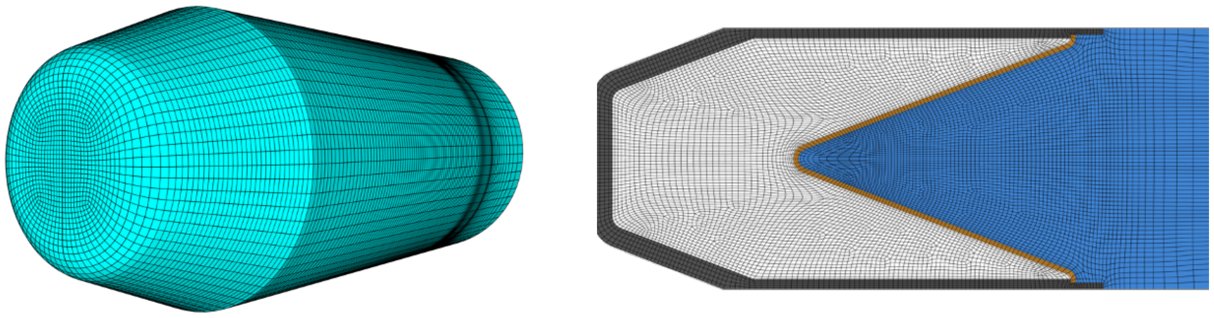

Figure 2-5: Example 1 million zone mesh of the shaped charge problem with materials identified by color.

under heavy development and not optimized, we were limited in mesh size by what was able to be generated and smoothed in a reasonable amount of time. This limited us to a zone of size about 64 million zones.

Furthermore, within the `Burton` specialization, there were constraints on data types that limited us to being able to read in only a mesh of size 16 million zones in the wallclock time of an allocation.[10] Our results for scaling then only go up to that scale as seen in the next section. The largest size mesh we tested on a single node was one of 8 million zones, or 1 million zones per GCD on RZVernal. We are fairly confident we can get larger than that within the memory limits on RZVernal after some optimization, but this is still over 24 million degrees of freedom per GCD. Of course, as we add more complicated physics and fields, the number of zones per GPU on this hardware will decrease.

3 Results and Discussion

The main technical result of this milestone is a scaling study of the shaped charge problem. We used meshes like those in Figure 2-5 of various scale as outlined above. The material models used are given in Table 3-1 where the colors correspond to those in Figure 2-5. The HE was detonated on the axis of symmetry where the steel case meets the HE. A detonation front starts from there and propagates towards the cone tip of the copper liner. Shocks reflect off the case, and ramp up the incline of the liner causing it to be compressed into a jet.

Material	Color	EOS	Strength	HE burn
steel	grey	Gruneisen	Steinberg-Guinan+melt	-
HE	white	JWL	-	direct lighting
copper	orange	Gruneisen	Steinberg-Guinan+melt	-
air	blue	γ-law	-	-

Table 3-1: Material properties used in the shaped charge calculation. Colors correspond to those in Figure 2-5.

[10]The naïve implementation used `int`s to index into arrays of the number of *bytes* we were sending via MPI, which caused an overflow once exceeding 2 GB. Additional issues involved sending too much information. These are resolved.

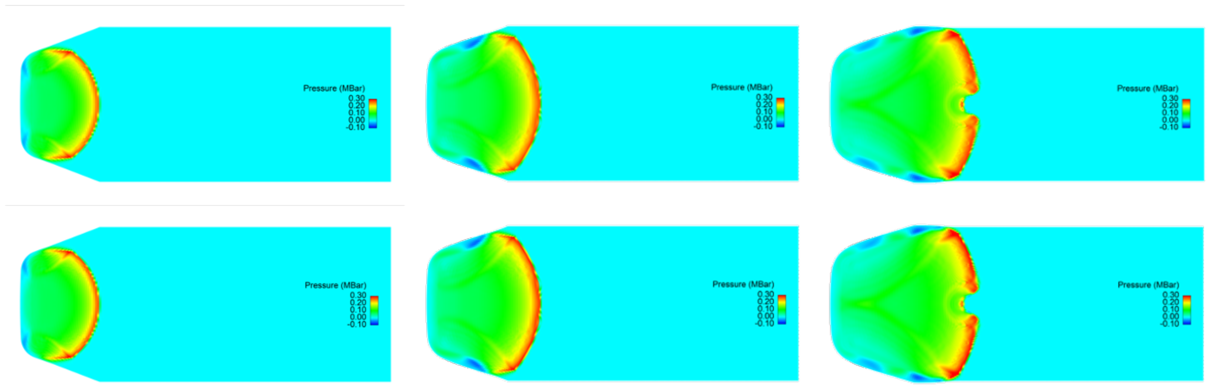

Figure 3-1: Two calculations of the shaped charge problem with Moya results on the top and FLAG results on the bottom. Each column is at a different time: 5μs, 7.5μs, and 10μs from left to right.

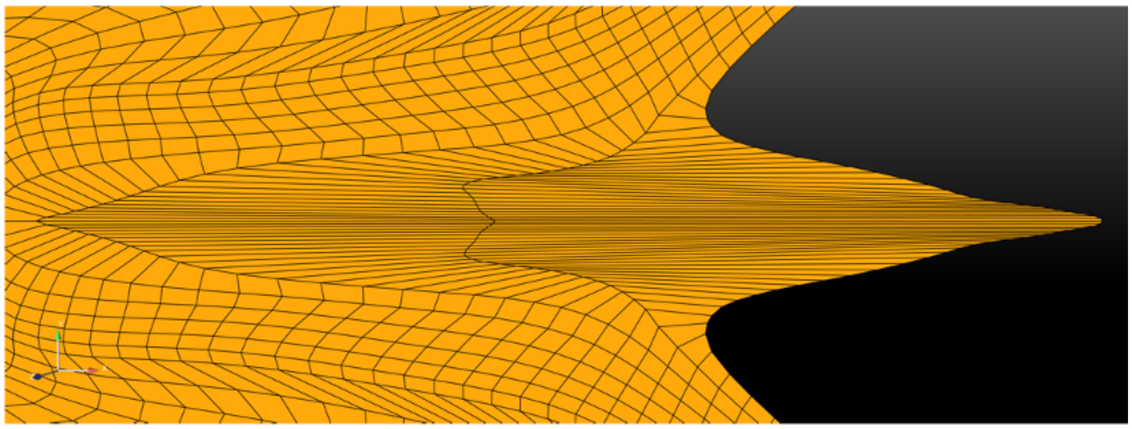

Figure 3-2: Close-up of the highly sheared zones with bad aspect ratio in the Moya calculation just before tangling.

Figure 3-1 shows typical pressure results of these calculations for both Moya (top row) and FLAG (bottom). The three columns correspond to the simulation at different times: $t = 5\mu$s, 7.5μs, and 10μs, from left to right. Both codes were set up to run with identical physics models, and their results agree up to this point. After about 10μs, the mesh begins to tangle in both codes due to bad aspect ratio zones, as shown in Figure 3-2. At this point, we stop the simulations. Note that because FLAG has a full ALE capability, it is able to continue the simulation past this point, but it is a non-trivial task in devising a robust ALE strategy. Also, as noted above, Moya is still implementing its ALE capability via Portage, but we hope to have more robust ALE techniques.

The scaling study ran these calculations on up to 16 nodes of RZVernal. Each node of RZVernal has four GPUs, each with two GCDs, which appear to the system as individual accelerators. As such, each node has eight accelerators. We ran in a mode where each accelerator was given a single MPI rank. There are certainly optimizations to explore in the space of optimal layout on this hardware, but no attempt was made for this milestone effort. In addition, for the CPU parts of our code, we did not make an attempt to thread those with OpenMP through Kokkos.

Figure 3-3 shows a spider plot containing both strong and weak scaling of Moya for this problem on RZVernal using the MPI backend of FleCSI. Each different color point represents a mesh at

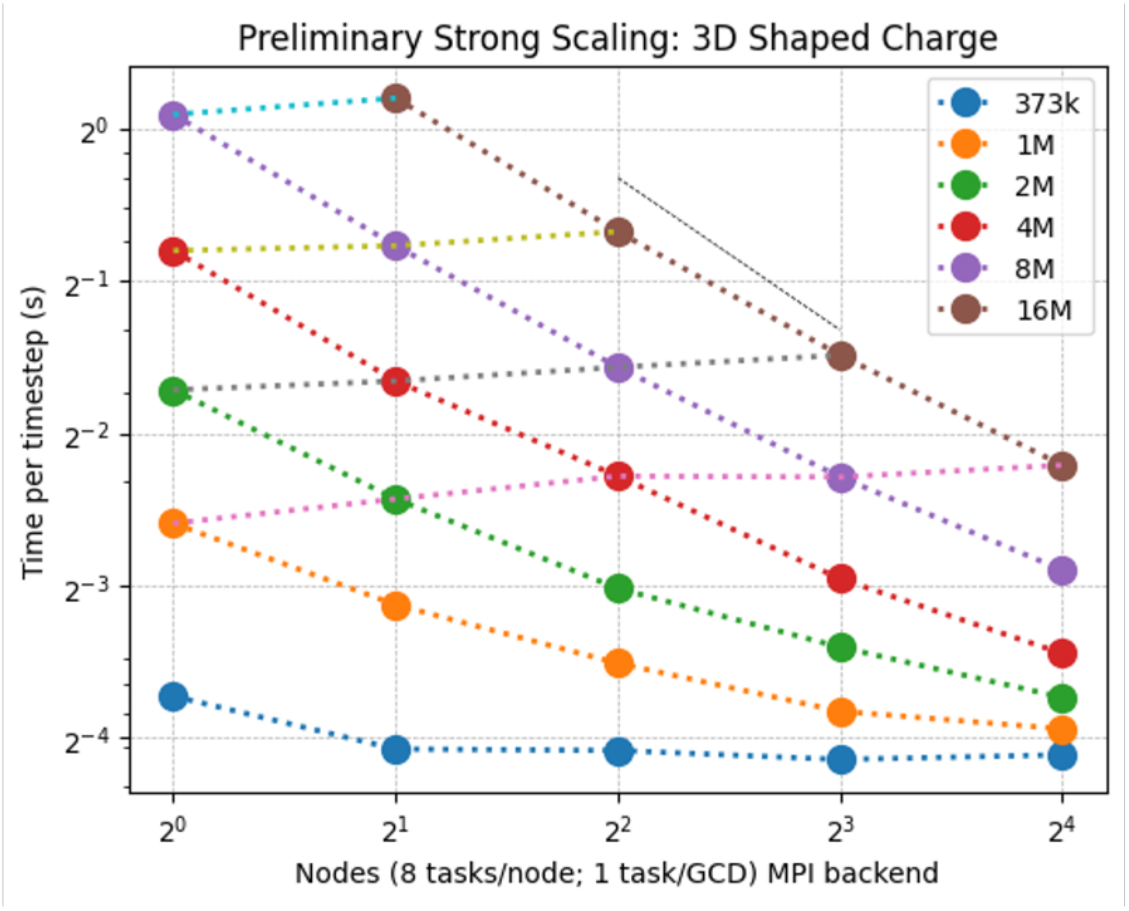

Figure 3-3: Spider plot of strong and weak scaling for Moya on the shaped charge problem on RZVernal. These results used the MPI backend of FleCSI.

a different size of zones. The lines that connect dots of the same color represent strong scaling. Clearly at small mesh sizes, the GPUs are not sufficiently taxed and the scaling is poor. As we increase in size, we approach the ideal scaling limit drawn as the thin dotted line. Again, there is still room for improvement, but our strong scaling is remarkably good given no optimizations. The lines that connect points of different color represent weak scaling. Ideal weak scaling would be identically flat; again our unoptimized results are reasonable. Figure 3-4 shows a comparison of performance between Moya and FLAG on CTS-1 and Moya on RZVernal (called EAS-3 here). In the calculations run on CTS-1 hardware, 36 ranks were used on the single node; the RZVernal calculations used eight ranks as described above. We find that (non-threaded) Moya is about 40% slower than FLAG on the CPU hardware of CTS-1, and about four times faster on the GPUs. Again, we stress that Moya has not been optimized in any way, and we expect to see much larger CPU-to-GPU comparisons in the future.

The scaling performance using the Legion backend is another story on this hardware. Figure 3-5 shows *strong* scaling for a Sod shock tube problem. This is a much simpler problem than the shaped charge problem described above, and yet the slope has opposite sign from what it should have in an ideal setting! It is worth keeping in mind that this was the first such FleCSI calculation

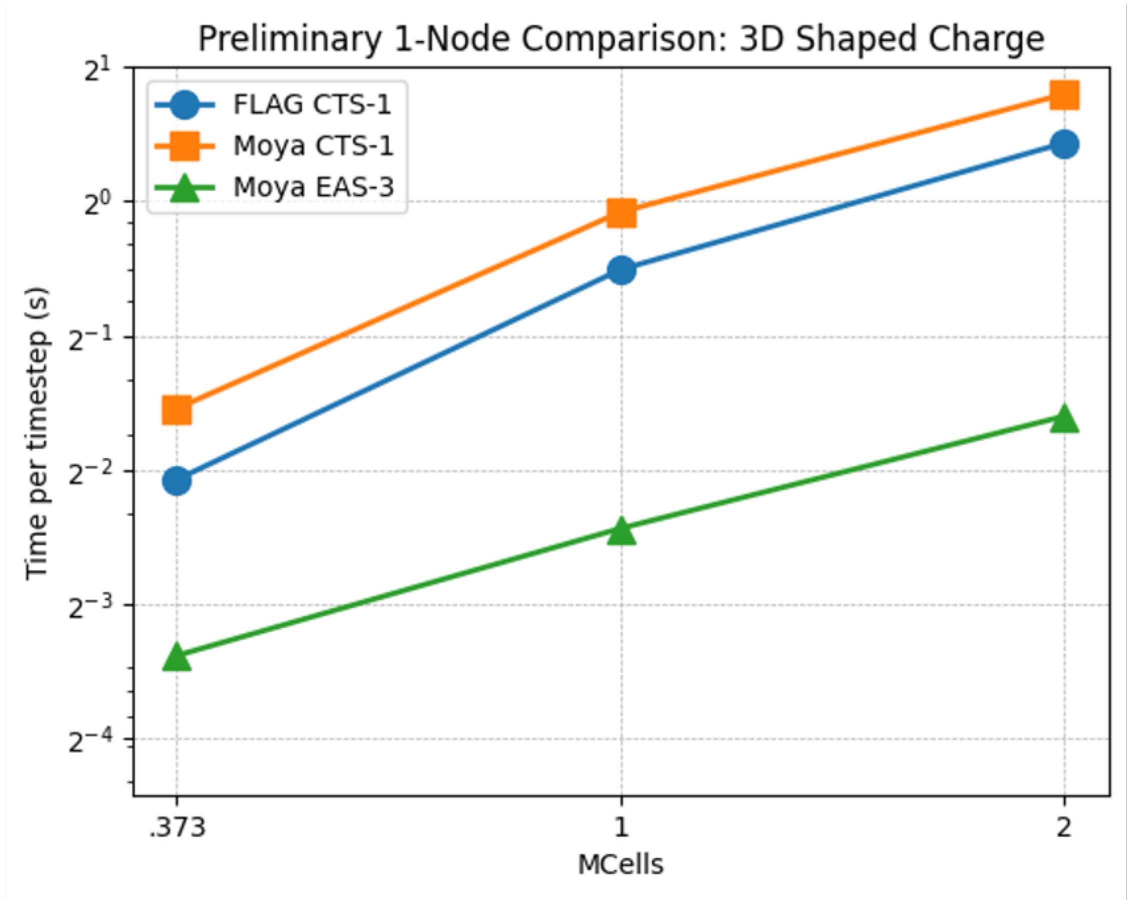

Figure 3-4: Comparisons of runtime per timestep between `Moya` and `FLAG` for the shaped charge problem. `FLAG` (circles) was run on CTS-1 hardware with 36 ranks. `Moya` was run on CTS-1 (squares) with 36 ranks and on RZVernal (triangles) with eight ranks.

on this hardware with the `Legion` backend. Since this calculation *some* improvement has been made, but we continue to work with the `FleCSI` and `Legion` teams on performance on this hardware.

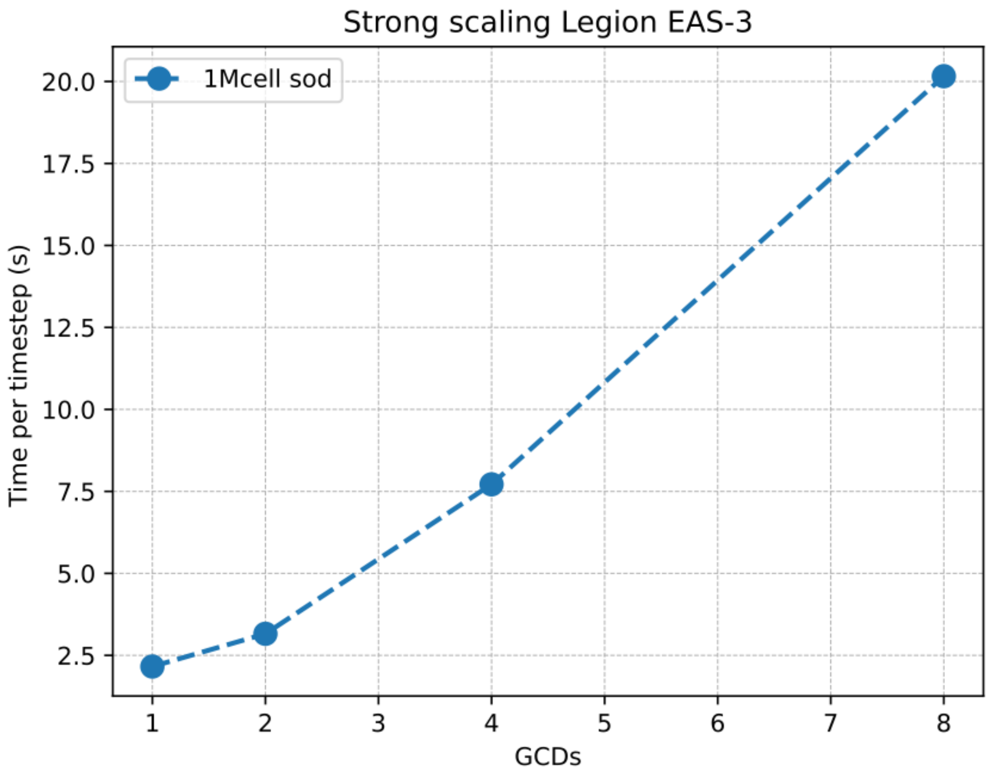

Figure 3-5: Strong scaling of a Sod shock tube problem using the `Legion` backend of `FleCSI` for the first time on RZVernal. Improvements have been made and continue to be made since this calculation.

4 Conclusion

This milestone set out to demonstrate readiness of ATDM codes to run mission-relevant calculations on El Capitan when it becomes available. To this end, we developed a new multiphysics code, `Moya`, which contained a minimal set of physics capabilities to run a three-dimensional shaped charge calculation. We leveraged the AMD GPU hardware on a pre-El Capitan system, RZVernal, and demonstrated that we could scale our shaped charge calculation out to half of the published system size.

We could only simulate until the start of jet formation within `Moya` due to our lack of ALE remap capability. Even so, we have the start of the capabilities required to run such calculations to completion, and we are actively working on getting `Portage` remap capability working within `Moya`. Shaped charge calculations have direct mission-relevance.

The hardware on RZVernal, while a precursor to El Capitan, is actually quite a bit different than that of El Capitan. On El Capitan proper, the memory access patterns for the GPU will be simplified, and because we aren't explicitly instantiating memory allocation calls on the device (because we are using `Kokkos`), our code should not need to change to run on El Capitan. As stated several times in this report, however, there is plenty of room for us to optimze for the hardware, when it arrives. One issue to keep in mind is that the memory *size* will be less on El Capitan, and we may have to be a bit more judicious about how we distribute our data. On RZVernal, we did not run into memory issues outside of having high register pressure as indicated from profiling results; this is something we will have to watch on El Capitan as well. Roofline plots indicate that we have somehwat low arithmetic intensity. This is to be expected from our type of hydrodynamics discretization, and with hydro being the main physics active during our shaped charge calculations. We will pursue avenuse to increase our arithmetic intensity, but we may be limited if we do not go to higher-order methods.

Follow-on work is to build out physics complexity, assess viability of `Legion`'s task-based parallelism for our methods, optimize our code, and port to other accelerated hardware, like that found in LANL's upcoming Venado machine. In addition, the Ristra project will be participating in a V&V L2 milestone in FY24. Of critical importance to the project as a whole is user adoption. To this end, we plan to tie into LANL's Common Modeling Framework, which is the entry point for users of LANL's multiphysics codes. This will allow those users to have a familiar workflow, but to run our codes. In FY24, we will start with some friendly users from the design community, and work to prioritize capabilities that will allow us to run problems of interest to that community.