Demonstration of Legion Runtime Using the PENNANT Mini-App

Charles Ferenbaugh
Los Alamos National Laboratory

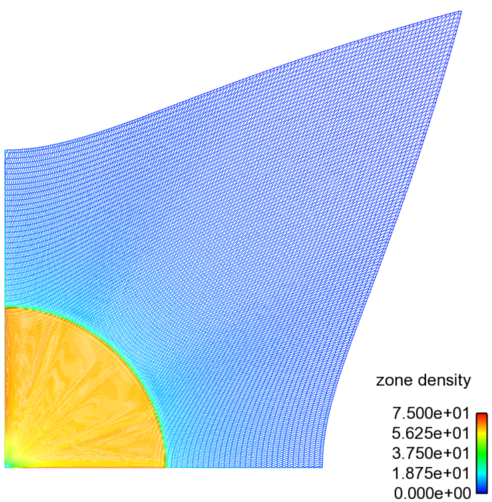
LA-UR-14-29180

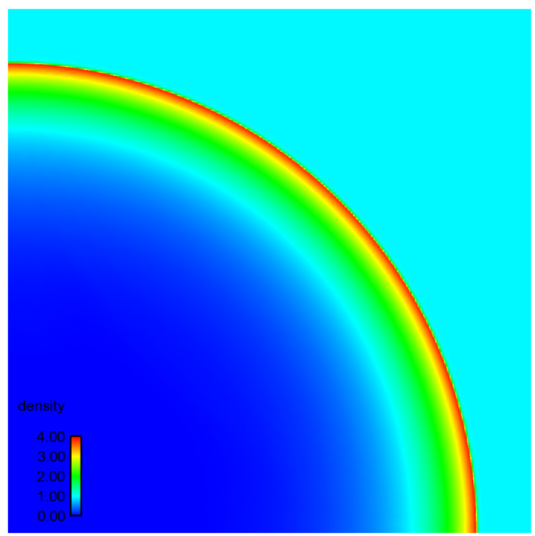
December 4, 2014

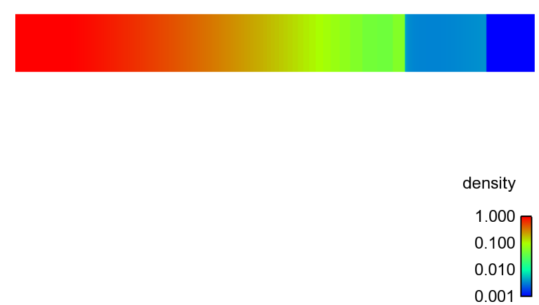
http://legion.stanford.edu

A brief overview of PENNANT

- **Implements a small subset of basic physics from the LANL rad-hydro code FLAG**
  - **Just enough to run a few standard test problems**







A brief overview of PENNANT (2)

- Operates on general unstructured meshes in 2D (arbitrary polygons, arbitrary connectivity)
  - This requires data structures, memory access patterns that don't occur in structured codes
- Contains about 3300 lines of C++ source code
  - Compare to > 600K lines for FLAG
- Has complete implementations for multicore CPUs (MPI + OpenMP) and GPUs (CUDA)
- Available open-source on GitHub

PENNANT implementation in Legion

- Legion version of PENNANT is code-complete
- Basic regression tests pass
- Some larger tests don't run yet
  - Still working on some bugs/unimplemented code paths in the Legion runtime
- Performance optimizations are in progress
  - Too early to show any performance numbers

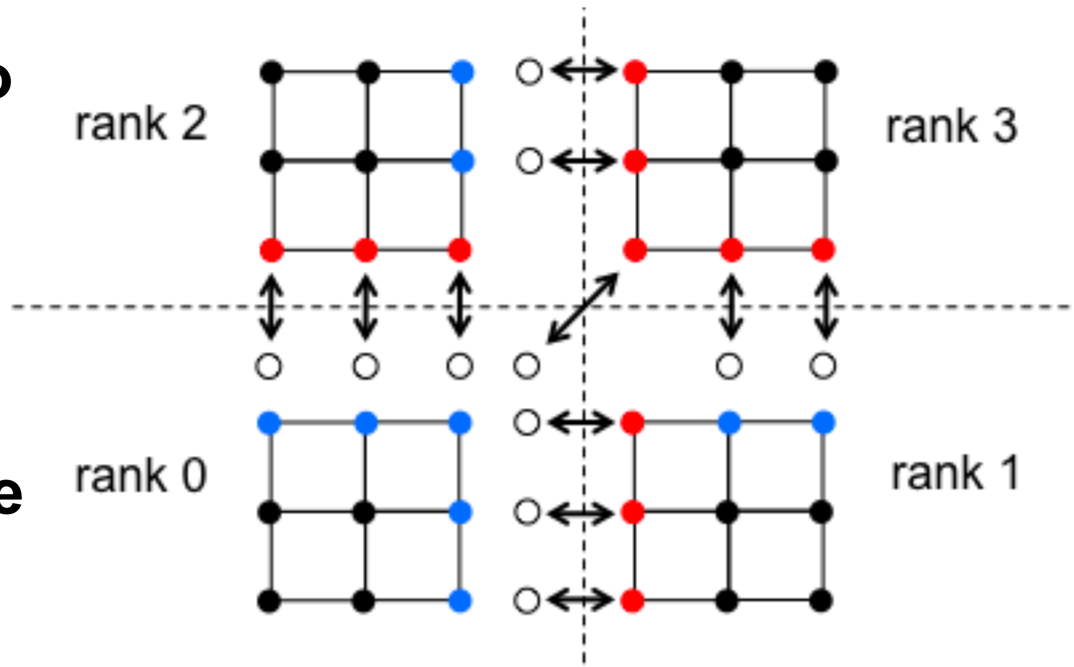
PENNANT implementation in Legion (2)

- There is a natural correspondence between:
  - MPI domain decomposition and Legion partitioning of index spaces
  - C++ function calls and Legion tasks

This makes PENNANT conversion to Legion straightforward (in principle)

PENNANT MPI parallelization
(similar to FLAG, and many other codes)

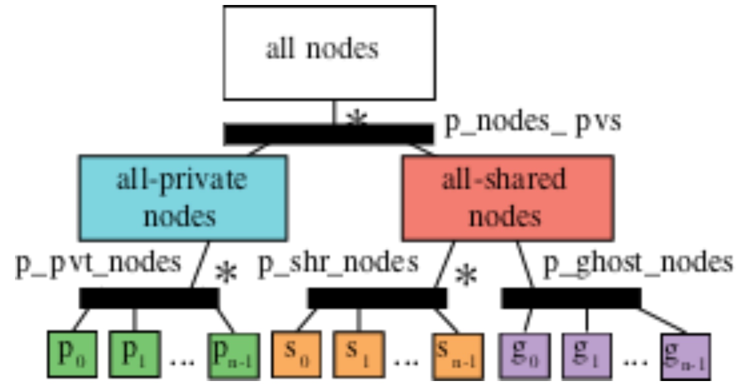
- Geometric domain decomposition onto MPI ranks
- Zones, sides, corners are private to each rank
- Boundary points are shared, duplicated between ranks; gather-sum-scatter implemented using MPI runtime calls

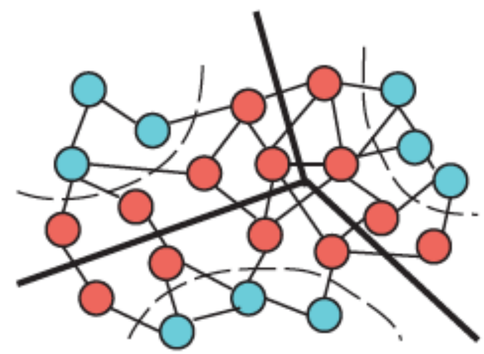


● = private, ● = master, ● = slave
○ = proxy (temporary, used for comms)

Legion partitions and data parallelism

- Legion can partition data into *private, shared,* and *ghost* partitions
- Legion can reason about dependencies, run tasks in parallel
- PENNANT MPI model can be expressed in this style
  - Explicit slaves, proxies no longer needed
  - Legion runtime will handle comms, data copies, synchronization





circuit graph example from the Legion SC12 paper

Example: function call in original

```
// calculate zone density
// zm     zone mass (input)
// zvol   zone volume (input)
// zr     zone density (output)
// zfirst, zlast  range of zone indices to compute (input)
calcRho(zm, zvol, zr, zfirst, zlast);
```

Example: function call in Legion

```
// create launcher for zone density task
IndexLauncher launchcr(TID_CALCRHO, dompc, ta, am);
// specify input fields over zones
launchcr.add_region_requirement(RegionRequirement(
        lpz, 0, READ_ONLY, EXCLUSIVE, lrz));
launchcr.add_field(0, FID_ZM);
launchcr.add_field(0, FID_ZVOL);
// specify output field over zones
launchcr.add_region_requirement(RegionRequirement(
        lpz, 0, WRITE_DISCARD, EXCLUSIVE, lrz));
launchcr.add_field(1, FID_ZR);
// launch task
runtime->execute_index_space(ctx, launchcr);
```

Comments on Legion porting

- The data model and function/task mapping made the port easy, in principle
  - Much of the work was repetitive, tedious
  - Working out the details was sometimes complicated
- PENNANT uncovered a number of bugs in the Legion runtime
  - The unstructured data capability in Legion hadn't been exercised as much as structured
- The process should be easier in the future
  - Abstractions, automation will help
  - Runtime will become more robust as it's exercised more

Summary

- A Legion version of PENNANT is working
  - Demonstrates that Legion can, in principle, support hydrocodes and similar LANL applications
- Data models and functions from current MPI codes can map naturally into the Legion model
  - Legion can then expose additional parallelism, provide new capabilities
- Current version of Legion is mostly working, but some aspects can be difficult to use
  - This should improve with time

Any questions?

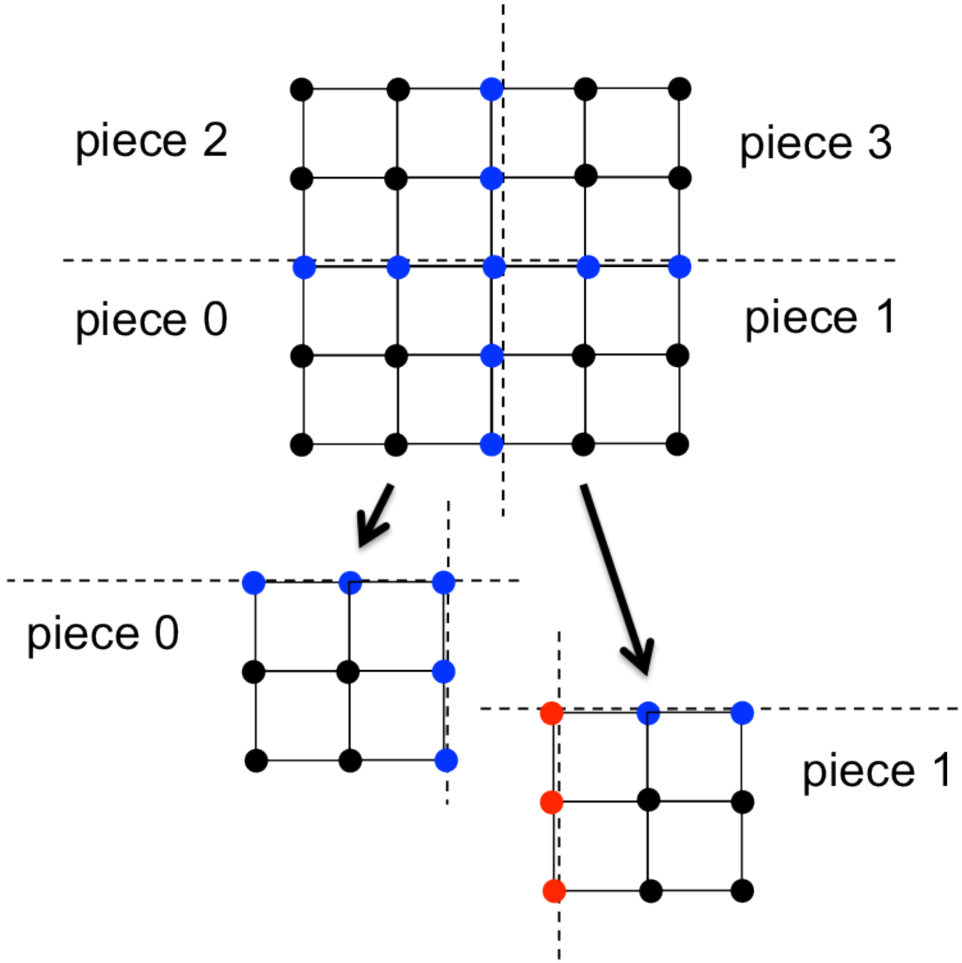
cferenba@lanl.gov

github.com/LosAlamos/PENNANT

Backup slides…

PENNANT Legion parallelization strategy

- **MPI decomposition can be translated into private/shared/ghost model**
  - **No duplicated points; slaves and proxies not needed**
  - **Boundary points visible to other pieces as “ghosts”**
- **Legion runtime handles comms, synchronization**



● = private, ● = shared, ● = ghost

Example: function body in original

```
// calculate zone density, given zone mass and volume
void Hydro::calcRho(
        const double* zm,
        const double* zvol,
        double* zr,
        const int zfirst,
        const int zlast) {

    #pragma ivdep
    for (int z = zfirst; z < zlast; ++z) {
        zr[z] = zm[z] / zvol[z];
    }
}
```

Example: function body in Legion (1 / 2)

```
void Hydro::calcRhoTask(
        const Task *task,
        const std::vector<PhysicalRegion> &regions,
        Context ctx,
        HighLevelRuntime *runtime) {

    // get field id's for fields to use
    FieldID fid_zm = task->regions[0].instance_fields[0];
    FieldID fid_zvol = task->regions[0].instance_fields[1];
    FieldID fid_zr = task->regions[1].instance_fields[0];
    // get accessors for fields
    RegionAccessor<AccessorType::Generic, double> acc_zm =
        regions[0].get_field_accessor(fid_zm).typeify<double>();
    RegionAccessor<AccessorType::Generic, double> acc_zvol =
        regions[0].get_field_accessor(fid_zvol).typeify<double>();
    RegionAccessor<AccessorType::Generic, double> acc_zr =
        regions[1].get_field_accessor(fid_zr).typeify<double>();
```

Example: function body in Legion (2 / 2)

```
    // loop over index space and compute densities
    const IndexSpace& isz =
            task->regions[0].region.get_index_space();
    for (Domain::DomainPointIterator itrz(isz); itrz; itrz++)
    {
        ptr_t z = itrz.p.get_index();
        double m = acc_zm.read(z);
        double v = acc_zvol.read(z);
        double r = m / v;
        acc_zr.write(z, r);
    }

}
```

What about task parallelism?

- Legion model fully supports task parallelism
- PENNANT has limited opportunities to use it
  - Single physics, single material
  - Explicit method, no iterative solvers
  - Load remains well-balanced through the run
- FLAG, and other LANL codes, would be better suited to exercising task parallelism