Progress Porting ALE3D to the GPU

Arlie Capps, Peter Robinson, Joseph Chavez {capps2, robinson96, chavez35}@llnl.gov

Lawrence Livermore National Laboratory

Summary

Porting ALE3D to the GPU requires foundational capabilities for software acceleration. The code size, age, and heavy use and development impose constraints on these facilities.

Capabilities	**Constraints**
Code execution	Portable
Data movement	Minimally intrusive
Performance monitoring	Easy to understand

We chose the RAJA library to enable execution on the GPU, CHAI to move data between host and device, and developed the SPOT tool for performance monitoring.

Design

Enable device acceleration *file-by-file* (incremental porting!)

RAJA — `https://github.com/LLNL/RAJA`

- Captures loop bodies as lambdas, variables by value
- Launches lambda as a CUDA kernel (can also run on CPU; OpenMP 4 backend in progress)

CHAI — `https://github.com/LLNL/CHAI`

- Copies data between host and device
- `ManagedArray<type>` object replaces bare `type *` in code, tracks host and device buffers
- Copy to host on raw pointer cast, to device when CUDA kernel starts

---

Main Tasks to port a file:

- Change all loops to our RAJA macro wrappers:
```
RAJA_LOOP_BEGIN (i, 0, n) {
// ...  loop body ...
} RAJA_LOOP_END
```
- Change all `int *`, `real8 *`, etc. to `int_ptr`, `real8_ptr` (which are `ManagedArray<int>`, `ManagedArray<real8>` typedef'd to more convenient name)
- At top of file, write
```
#define GPU_ACTIVE
```

Then address remaining issues, including:

- Move log and error I/O out of RAJA loops
- Refactor loop counting and summing operations to use RAJA reduction types

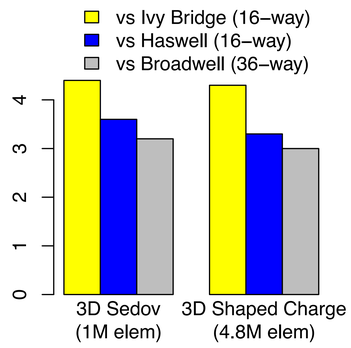
Speedup

Speedup:
$$\frac{\text{walltime, 1 production node}}{\text{walltime, 1 Sierra preview node}}$$

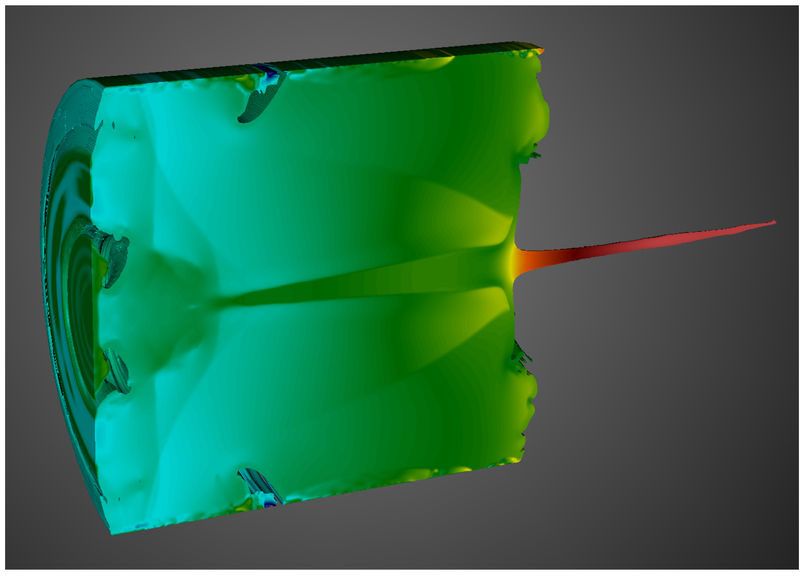
Sierra preview node uses one of 20 POWER8+ **cores** with one of four P100 GPUs (single-threaded).

Production nodes and speedups shown in bar chart.

Main focus has been advection and Lagrange; MPI is the **next major effort**.

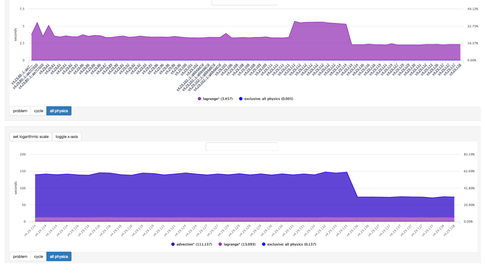


Shaped Charge on GPU



Performance Monitoring

We track timing hierarchy for all builds and problems over time. **SPOT**, a web application, lets us drill down to see what's slow.



Demo on demand!

Pain points, Wins

Loop and pointer type replacement touches the entire code

- No way around an enormous amount of work
- Tools (text and GUI) can automate some of the process and help with consistency

RAJA macros are flexible

- variants for iterating over faces, nodes, other data structures
- `RAJA_SCAN` variant easily added; large performance gain
- More than one backend (CPU, CUDA to GPU, soon OpenMP)
- Macro, so we can print code context on error

CHAI abstraction is powerful

- Data copy (ONLY) on touch (cast to pointer)
- This can be foiled by pointer aliasing
- When all else fails, dump all data on transfer
- Because CHAI is an abstraction, we plan to use UM backend when it's available and copying when it isn't

Warts

- Capture by value: `this` pointer is a *host* address, use on device will segfault; virtual functions have similar problem
  - `this` capture (CUDA/C++17) may help
  - vtable idiom currently under development
- Some huge loops, some tiny loops
  - Loop fission can sometimes help register spilling
  - Loop fusion or persistent kernels may help launch overhead
- Some loops need (extensive) rewriting in other ways
  - Adopt scan idiom
  - Calls into GPU-unfriendly libraries (few of these so far)

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-POST-736959