AMReX in 2020: Porting for Performance to GPGPU Systems

2020 Performance, Portability, and Productivity in HPC Forum

Kevin Gott, Andrew Myers and Weiqun Zhang
Lawrence Berkeley National Laboratory
Sept 1, 2020
AMReX: Block-Structured AMR Co-Design Center

- **Mesh, Particle, AMR, Linear Solvers, Cut-Cell Embedded Boundary**
- Written primarily in **C++**, with Fortran interfaces
- **MPI + X**
  -- OpenMP on CPU
  -- CUDA, HIP, DPC++ internally on GPU
- Solution of parabolic and elliptic systems using **geometric multigrid solvers**.
- Support for multiple **load balancing** strategies.
- HDF5 and native I/O formatting supported by Visit, Paraview, yt and Amrvis.
AMReX Across Science

Combustion (Pele)

Astrophysics (Castro)

Cosmology (Nyx)

Other applications:
- Phase field models
- Microfluids
- Ionic liquids
- Non-Newtonian flow
- Fluid-structure interaction
- Shock physics
- Cellular automata
- Low Mach number astrophysics
- Defense science

Multiphase flow (MFIX-Exa)

Exawind
Exascale Strategy

• **Port with native models: (CUDA/HIP/DPC++).**
  o Vendor API calls wrapped in AMReX functions. User doesn’t need to directly use them (or pragma methods), but still can, if desired.

• **Encourage users to move to C++.**
  o Users may utilize Fortran interfaces, but development focus will be C++.

• **Seek and implement portable performance solutions.**
  o Native, C++ based solutions are preferable.

• **Leverage available vendor support and user expertise.**
  o Remain open to all improvements and observations to maximize performance at launch.
Porting to Exascale
Overview of GPU Porting Strategy

• Native CUDA/HIP/DPC++ implementation.
  o Begin with focus on matching software and hardware. Cross-platform performance and implementations will be studied when interest arises.

• First design pass based on CUDA implementation.
  o Streams, managed memory, minimize data movement, etc.

• Working directly with engineers from both AMD and Intel.
  o Testing, discussing and improving the new compilers, libraries and required features.
## Overall Porting Progress

<table>
<thead>
<tr>
<th>CUDA</th>
<th>HIP</th>
<th>DPC++</th>
</tr>
</thead>
<tbody>
<tr>
<td>● CUDA port <strong>completed</strong> and successful.</td>
<td>● HCC port was successful, but limited by compiler features.</td>
<td>● DPC++ port actively <strong>underway</strong>. Regular Zoom meetings with Intel and Argonne engineers.</td>
</tr>
<tr>
<td>● Most ECP applications are <strong>finishing port or beginning performance development</strong>.</td>
<td>● HIP-clang port is underway, working with HIP and HPE engineers through email and Confluence.</td>
<td>● Developing on Iris and Intel’s DevCloud.</td>
</tr>
<tr>
<td>● Actively conducting portable performance studies in CUDA, based on CUDA paradigm.</td>
<td>● Developing primarily on Tulip.</td>
<td>● Good progress made, with numerous issues and feature requests investigated.</td>
</tr>
<tr>
<td></td>
<td>● Once compiler is ready, expect a smooth transition, given target is CUDA 8 and our previous HCC experience.</td>
<td>● <strong>Performance</strong> to be investigated soon.</td>
</tr>
</tbody>
</table>
Porting Status

Overview of the **current feature requests and issues** for HIP and DPC++ is available in the AMReX repo:

- [https://github.com/AMReX-Codes/amrex/tree/development/Docs/Notes/HIPIssues.md](https://github.com/AMReX-Codes/amrex/tree/development/Docs/Notes/HIPIssues.md)
- [https://github.com/AMReX-Codes/amrex/tree/development/Docs/Notes/DPCPPWishlist.md](https://github.com/AMReX-Codes/amrex/tree/development/Docs/Notes/DPCPPWishlist.md)

Relevant **regression testing** is also underway:

- [https://github.com/WeiqunZhang/amrex-gpu-status/](https://github.com/WeiqunZhang/amrex-gpu-status/)

Recent talk on **progress with Intel**: “Early Experiences Supporting DPC++ in AMReX”:


Overall, **very pleased** with the support being given by AMD and Intel.
Portable Performance Development
Performance Strategy

5 Categories of Performance Development:

1) Maximize FLOPs for Floating Point Work. (GPU)

2) Improve Metadata performance. (CPU + OpenMP)

3) Minimize / Improve Comms.

4) Obfuscate I/O Operations.

5) Extend Parallelism wherever possible.

Ongoing by AMReX developers, app teams and users.

Async I/O is implemented and available.

 Longer-term strategies under initial investigation.
Fused Launches

Added mechanism to **fuse small kernels** to reduce launch overhead.

- Based on RAJA’s implementation.
- Moves selected **device lambdas to GPU** to be ran together.
- At least **20% speedup** for $32^3$ or smaller boxes.
- Working on **fully automated** implementation.

```cpp
Gpu::FuseSafeGuard fsg(true);
for (MFIter mfi(mfc); mfi.isValid(); ++mfi) {
    if (fuse_this_kernel(mfi)) {
        amrex::Gpu::Register(vbx,
            [=] AMREX_GPU_DEVICE (int i, int j, int k) {
                <Kernel contents>
            });
    } else { <launch normally> }
}
amrex::Gpu::LaunchFusedKernels();
```

Turn fusing on.

Register a kernel to be fused.

Launch the chosen fused kernels.
Particle-Mesh operations on V100

Particle-mesh operations benefit from periodic sub-grid sorting to take advantage of memory hierarchy.

- > 6x speedup over unsorted deposition / gather.
- Flexibility in tile size and sorting interval allows tuning.
- Exploring ML approaches to choosing optimized parameters for different regimes.
GPU Load Balancing

Developing **accurate runtime** GPU kernel timers.

- Researching both **CUPTI** and native timers for load balancing of GPU-based simulations.

- GPU-side timing has proven very efficient for load balancing many AMReX cases.
Async I/O

Hides I/O time by writing to file on a copy of data while **work continues asynchronously**.

- Primarily designed for **GPU runs**, but **can also benefit CPU**.
- Targeted method for **all exascale systems**.
- Also being **developed by HDF5**.
- Requires **scalable MPI_THREAD_MULTIPLE**.

Speed up for print using different I/O messaging strategies when asynchronously overlapped with repeated Min and Max calculations (MPI_Reduce). (2 Threads).
Next Steps / Upcoming Performance

Primary focus: Continue HIP and DPC++ porting.

Performance issues under consideration:

• Memory pool with defragmentation.
• CPU + GPU asynchronous work across grids.
• Coarse-grained parallelization identification.
• AMReX-specific MPI regression testing.
Thank you for listening!
AMReX and its applications are using roofline analyses to get absolute performance measurements on GPU systems.

- Especially useful to find complex kernels in applications that should be targeted for study.
- **Nsight Compute** has proven popular and effective for quick analyses.
Assigning and moving particles to the proper level, grid, and MPI rank is one of the main parallel communication patterns in AMReX applications.

- Algorithm runs completely on GPU, taking advantage of AMReX parallel prefix sum implementation.
- Weak scaling to full Summit, with or without mesh refinement.