GRTeclyn updates (part 1), benchmarking and how to run on GPUs

Miren Radia

GRTL Collaboration Meeting

Wed 26 Jun 2024

GRTeclyn updates

Unit Tests: Catch2 → doctest

- Unit testing framework has been switched from Catch2 to doctest
- Syntax almost the same but there are some changes to make tests work with HIP (AMD GPUs) and SYCL (Intel GPUs).
- In MyTestCase.cpp, define a `void` function (declared in MyTestCase.hpp)

```
void run_my_test_case()
{
    <test code>
    CHECK(<condition that is true if test passes and false if test fail>);
}
```

- In TestCases.hpp call this function:

```
TEST_CASE("<test case name>")
{
  run_my_test_case();
}
```

- There is a README on how to add new tests.
- There is still a problem on Intel GPUs when built without `DEBUG = TRUE`.

Continuous Integration (CI)

- We now have a CI pipeline which builds and *runs* the unit tests on Nvidia GPUs.
  - Uses GitLab CI/CD (not GitHub Actions) via mirroring to Cambridge GitLab instance
  - Self-hosted GitLab runner running on Cambridge RCS cloud virtual machine (VM)
  - VM SSHs to CSD3 where it builds on the login node and then submits to the `ampere` queue to run on an A100.
- Some changes to the "Lint" GitHub action which checks for style/programming errors:
  - Will now fail if there are warnings.
  - Will write a nicely formatted list of warnings in a comment on a pull request.
  - Will ignore source files that are in the `.lint-ignore` file (e.g. files not yet ported to AMReX).

Derived variables refactoring (WIP)

- There is an open PR to make the following changes (should be merged soon):
  - Change of variable type terminology from GRChombo:
    "evolution" → "state", "diagnostic" → "derived"
  - Port Weyl4 from GRChombo and add in a test for its calculation
    - Test compares HDF5 outputs from GRTeclyn and GRChombo.
    - Need to build with `USE_HDF5 = TRUE` which is `FALSE` by default.
  - There is also a similar test for the constraints.
  - Unlike GRChombo, there are no fixed `MultiFab`s for derived quantities. They are created on the fly as they are needed.
  - Can calculate derived quantities on `MultiFab`s with ghost cells (required for AMR Interpolator).
    - Fills extra ghost cells in the source rather than in derived `MultiFab` (latter is what GRChombo does).
  - Switches from variable parameter parities to hardcoded variable parities.

How to run on GPUs

Building on Nvidia or AMD GPUs

- To run on Nvidia GPUs, AMReX uses CUDA.
  - This is the most mature AMReX GPU backend so is likely to be the most robust/performant.
  - DiRAC currently has some Nvidia GPU systems but procuring new ones is very difficult so next generation systems might not have them.
- To run on AMD GPUs, AMReX uses HIP.
  - The HIP programming model is very similar to CUDA but the software support is less mature (although continuously improving).
  - DiRAC are considering AMD GPUs for next generation systems.
- Load CUDA/HIP & GPU-aware MPI modules and build using the following

```
make -j <num jobs> USE_[CUDA/HIP]=TRUE AMREX_[CUDA/AMD]_ARCH=<target gpu arch>
```

  where `<target gpu arch>` is `80` for the Nvidia A100 or `gfx90a` for AMD MI200 series

Building/running on Intel GPUs/Dawn

- Building on Intel GPUs (just PVCs for now) more complicated. To run on Dawn:

```
USE_SYCL=TRUE SYCL_AOT=TRUE AMREX_INTEL_ARCH=pvc SYCL_PARALLEL_LINK_JOBS=24
```

- There is high register pressure so increase size of registers to improve performance:

```
SYCL_SUB_GROUP_SIZE=16 SYCL_AOT_GRF_MODE=Large
```

- Intel PVCs are comprised of 2 stacks (or tiles) so best to run 2 MPI ranks per GPU and pin each rank to a stack/tile.
- There are several `I_MPI_...` environment variables to set in order to use GPU-aware MPI and GPU pinning with Intel MPI
- Unfortunately with the latest oneAPI release on Dawn (2024.1), there are issues with MPI so stick with the previous version.
- Further details can be found in #67.
- Unit tests won't work unless `DEBUG = TRUE` (see #46).

General tips for running on GPUs

- There is more floating point performance in a single GPU than on a CPU node
  - e.g. ~10 TFLOPS theoretical FP64 peak on a single Nvidia A100 vs ~7 TFLOPS theoretical FP64 peak on a whole dual-socket Intel Ice Lake CPU node.
  - Simulations will need significantly fewer nodes than before.
- Best to use 1 MPI process per GPU (or tile/GCD in the Intel/AMD case) so much fewer than for CPU simulations.
- Increase box sizes to improve GPU occupancy. Fewer big boxes will be more efficient than lots of small boxes (though there is a trade-off with algorithmic efficiency).
- Read cluster documentation on pinning GPUs to specific MPI processes and network interface cards (NICs) - might need wrapper script:

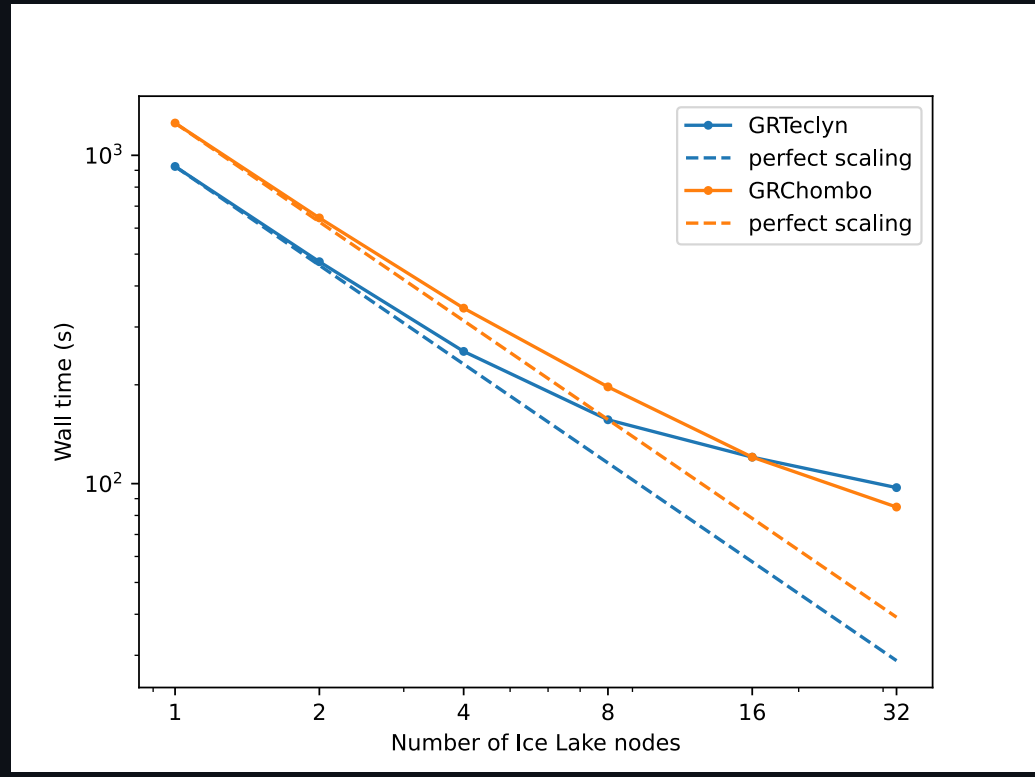
```
mpiexec <mpi options> ./gpu-pin-wrapper-script.sh <application> <options>
```

- Set param `amrex.use_gpu_aware_mpi = 1` to use GPU-aware MPI (~20% faster).

Benchmarking

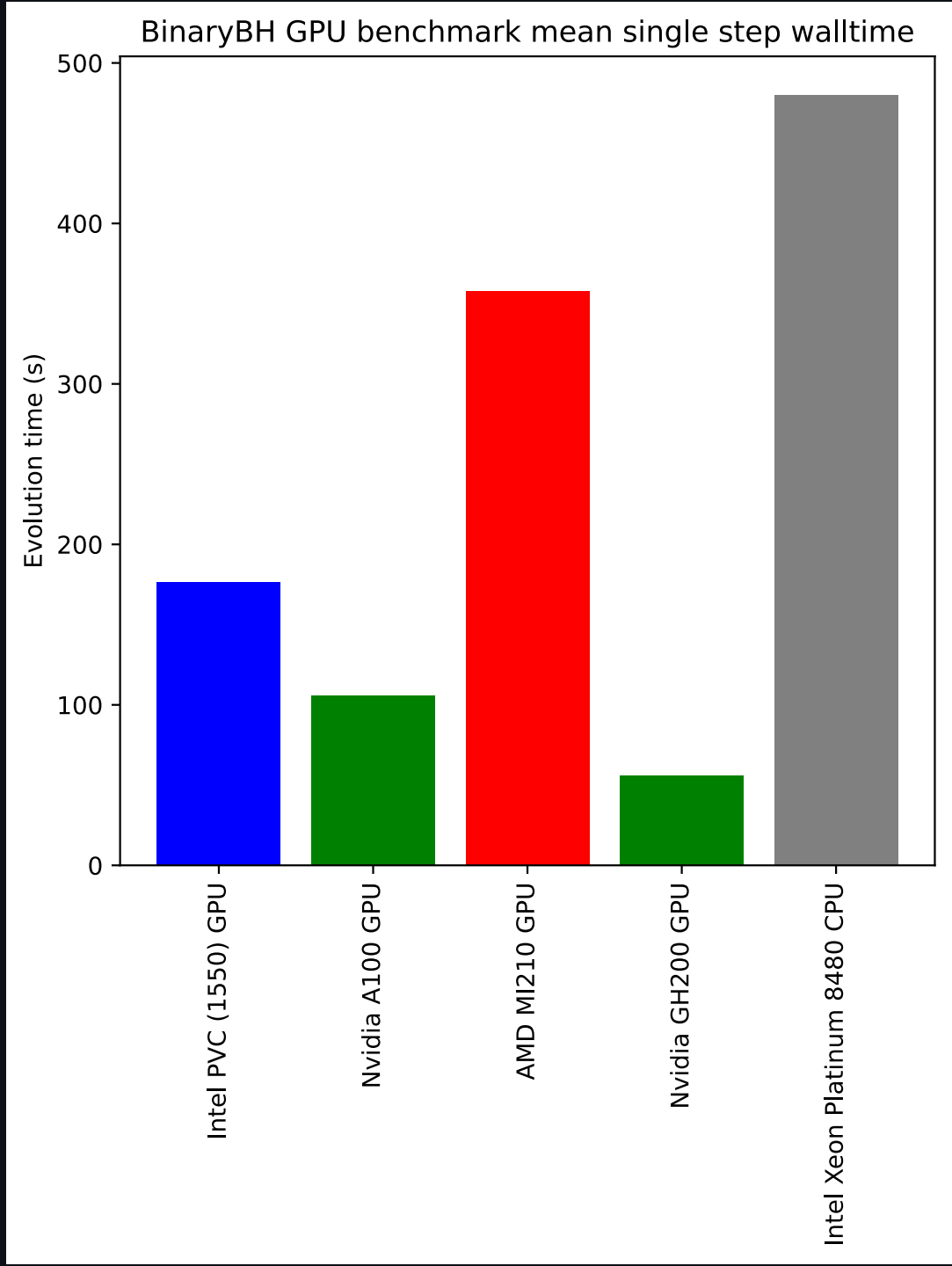
CPU strong scaling test (GRChombo vs GRTeclyn)

- Run on the CSD3 Ice Lakes with 4 OpenMP threads per MPI task.
- 128^3 grid on the coarsest level with 9 levels of refinement, $\Delta x_{\text{finest}} = M/64$
- 2 timesteps on the coarsest level
- GRChombo refines a bit more than GRTeclyn with the same tagging criterion and threshold
  - Slower for lower number of nodes but strong scales a bit longer
- Details in #59
- Should try re-running with a bigger problem to show stronger scaling



GPU benchmark comparison

- Running my BinaryBH GPU benchmark which is similar to the scaling test but with larger box sizes.
- Some caveats/notes
  - 1 MPI rank on Nvidia and AMD GPUs
  - 2 MPI ranks on PVC (1 per stack)
  - 28 MPI ranks with 4 OpenMP threads per rank on Sapphire Rapids CPU
  - AMD MI210 $\approx \frac{1}{2}$ AMD MI250 so less fair comparison



Any questions?