OVERFLOW on GPUs

Progress and Lessons Learned

Chip Jackson <charles.w.jackson@nasa.gov>
NASA Langley

Overset Grid Symposium 2022

Overview

- Goals of OVERFLOW GPU Port
- Miniapps - Lessons Learned
- Current Status
- What is Coming

This is a work in progress

Goals of OVERFLOW GPU Port

- Learn about GPU programming
- Determine if running on the GPU makes sense for OVERFLOW
- Have one path through the code fully ported to GPUs

- Long term: have feature parity between the CPU and GPU path in a sustainable way

Benefits of GPUs over CPUs

- Higher memory bandwidth (900-1500 GB/s vs. 50-150 GB/s)
  - Great for memory bound codes like OVERFLOW
- Expose much more parallelism for more performance (10-300 TFLOPS vs. 7-250 GFLOPS)
- More power efficient (FLOPS/W) than CPUs
- More space-efficient than CPU only hardware (can fit more compute in a single rack)

Downsides of GPUs

- Less flexibility than CPUs
- You have to have enough parallelism to fill the GPU, O(100,000)
- Data transfer between the CPU and GPU can be relatively slow
- Getting peak performance out of GPU can be difficult for complex problems
- You have to write your code specifically for the GPUs

How to write code for GPUs?

- Many different options
  - Vendor Specific: CUDA, HIP
  - Directive Based: OpenACC, OpenMP
  - Frameworks: Kokkos, RAJA, OCCA, OneAPI
- We selected OpenACC for it’s ease of implementation and native Fortran support but are aware of downsides to this approach
  - Do have some CUDA Fortran
  - Other groups have switched to C++ where you have a wider range of options for porting

Miniapps

Miniapps

- These have been critical to our success on both the CPU and GPU
- Simple, **small** codes that can be iterated quickly (build and run)
- Exhibit common motifs from application with a “correct” answer
- We were able to release these miniapps as open-source for easier collaboration with external partners
- We have created two miniapps

Central Solver Miniapp

Overview

- Form the 2nd-order Euler residual with the central scheme (F3D smoothing)
- Form the batch scalar-pentadiagonal LHS
- Solve the batch scalar-pentadiagonal scheme

Motifs

- Stencil operations common throughout OVERFLOW
- Scalar-pentadiagonal build and solve

Central Solver Miniapp

Lessons Learned

- Several changes were required to get good performance out of the miniapp
  - Expose more parallelism
  - Do more work in a kernel
  - Hide launch latency with `ASYNC(stream)` clauses
- Unfortunately this means code is not single-source
  - Changes should help CPU but break auto-vectorization

Central Solver Miniapp

Example

```
!$acc parallel loop gang collapse(2) async    &
!$acc    present(batch,grids,solns,sources)   &
!$acc    private(ig)
do iig = 1, batch%ngrids
  do l   = 1, batch%lmax
    ig = batch%ig_map(iig)
    if( l <= grid(ig)%ld ) then

!$acc loop vector collapse(2)
      do k = 1, grid(ig)%kd
      do j = 1, grid(ig)%jd
        ! do something
      end do
      end do

    end if
  end do
end do
```

Chimera Boundary Exchange Miniapp

Overview

- Interpolate solution
- Pack up interpolated solution
- Transfer buffers
- Unpack buffers

- Not a significant portion of runtime on CPUs, but rather a technical barrier to getting OVERFLOW running on GPUs

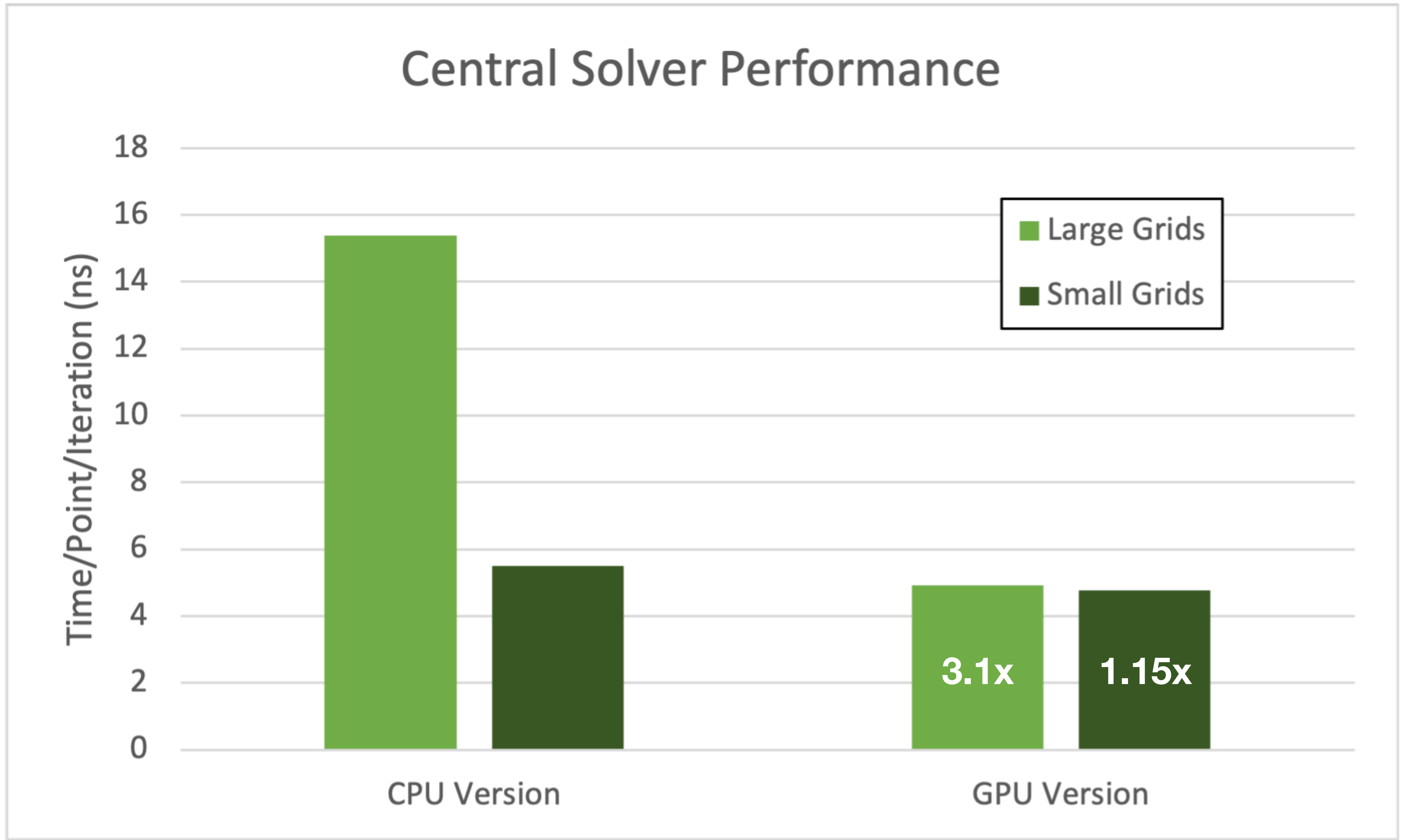
Chimera Boundary Exchange Miniapp

Lessons Learned

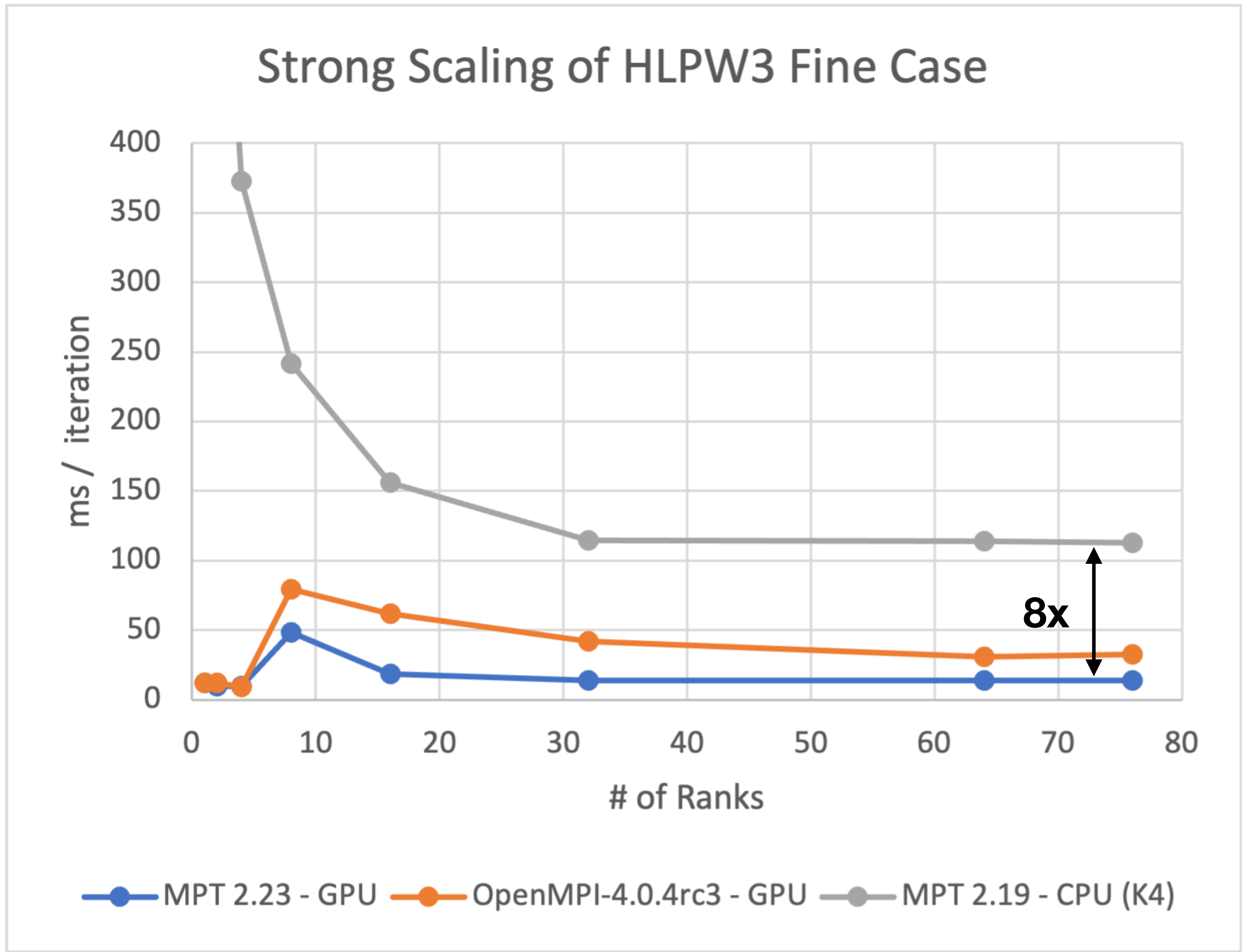
- We are using 1 MPI rank per GPU
- Easily and efficiently move data between GPUs with OpenACC and CUDA-aware MPI

```
!$acc host_data use_device(buff) if_present
      call MPI_Send( buff, count, my_mpi_real, dest, tag, comm, ierr )
!$acc end host_data
```

Latest Status of the OVERFLOW Miniapps



Single V100S GPU, Dual Socket Intel Gold 6148 Skylake CPU node



V100 GPU at NAS (1 rank / GPU) vs.
Dual Socket Intel Gold 6148 Skylake CPU at LaRC (1 rank / Core)

Porting the Full Application

Current Status

- Pulled the work from the miniapps into the full application
- Read problem inputs on the CPU then branch off for the GPU path

1. Ensure the requested options have been ported
2. Transfer the necessary data to the GPU
3. Run the solver on the GPU
4. Transfer data back to the CPU for output and post-processing

Ported Options

- Only have a small subset of the capabilities ported to the GPUs so far (based on the miniapps)
  - Central scheme, scalar-pentadiagonal solver, SA-neg-noft2 turbulence model, certain boundary conditions, no grid sequencing/multi-grid, static grid systems
- We do extensive option checking to ensure that the requested path is supported
- Still a work in progress to add more features and improve the performance

Future Plans

- Add upwinding schemes (Roe, HLLE++)
- Add moving grid capability
- Add SSOR path

- Other features that users want first?

Questions? Comments?

Chip Jackson
charles.w.jackson@nasa.gov