

Porting OVERFLOW CFD Code to GPUs

To Hackathons and Beyond!

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

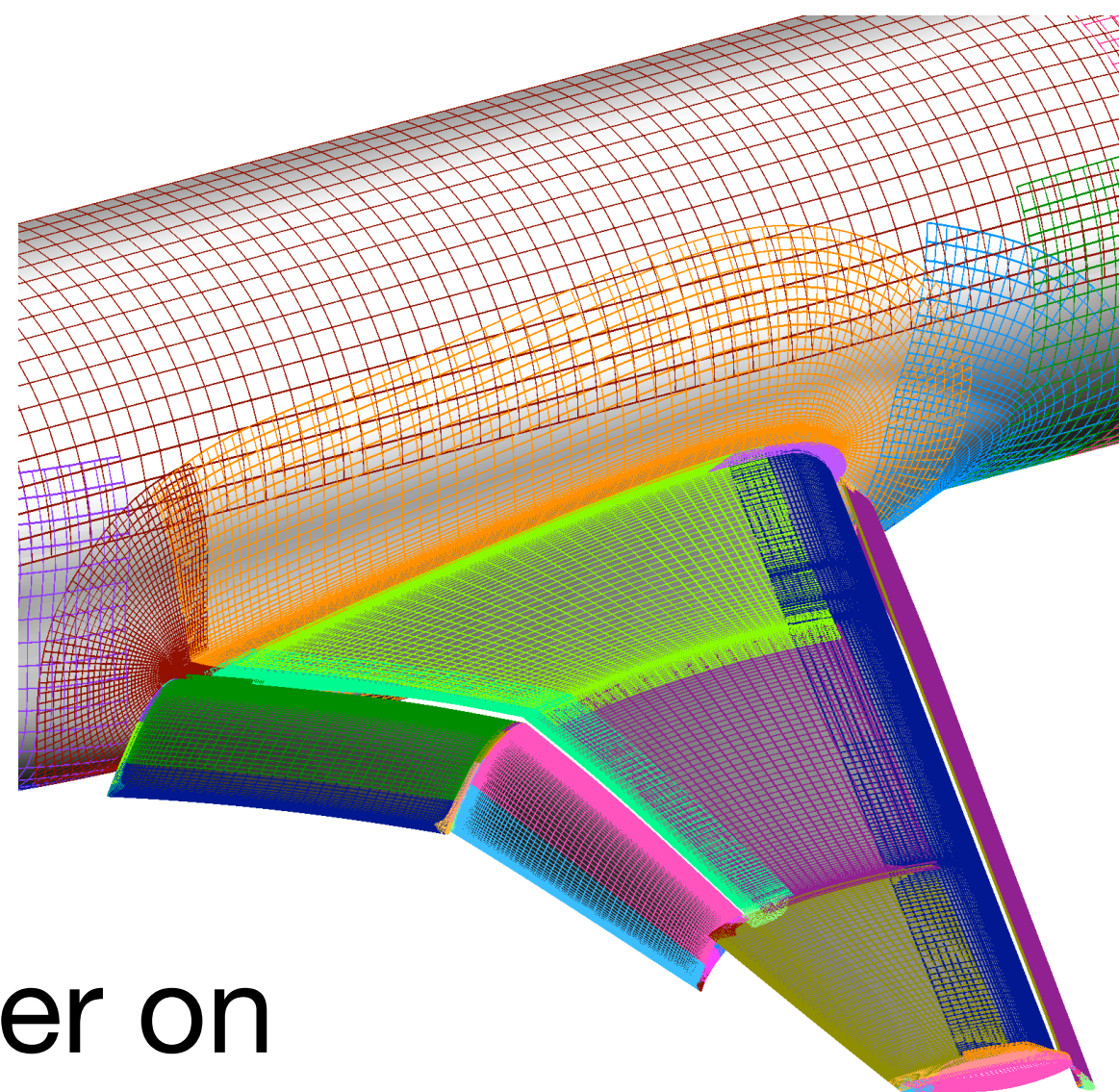
NASA Langley Research Center

OpenACC and Hackathons Summit 2022 — August 4, 2022

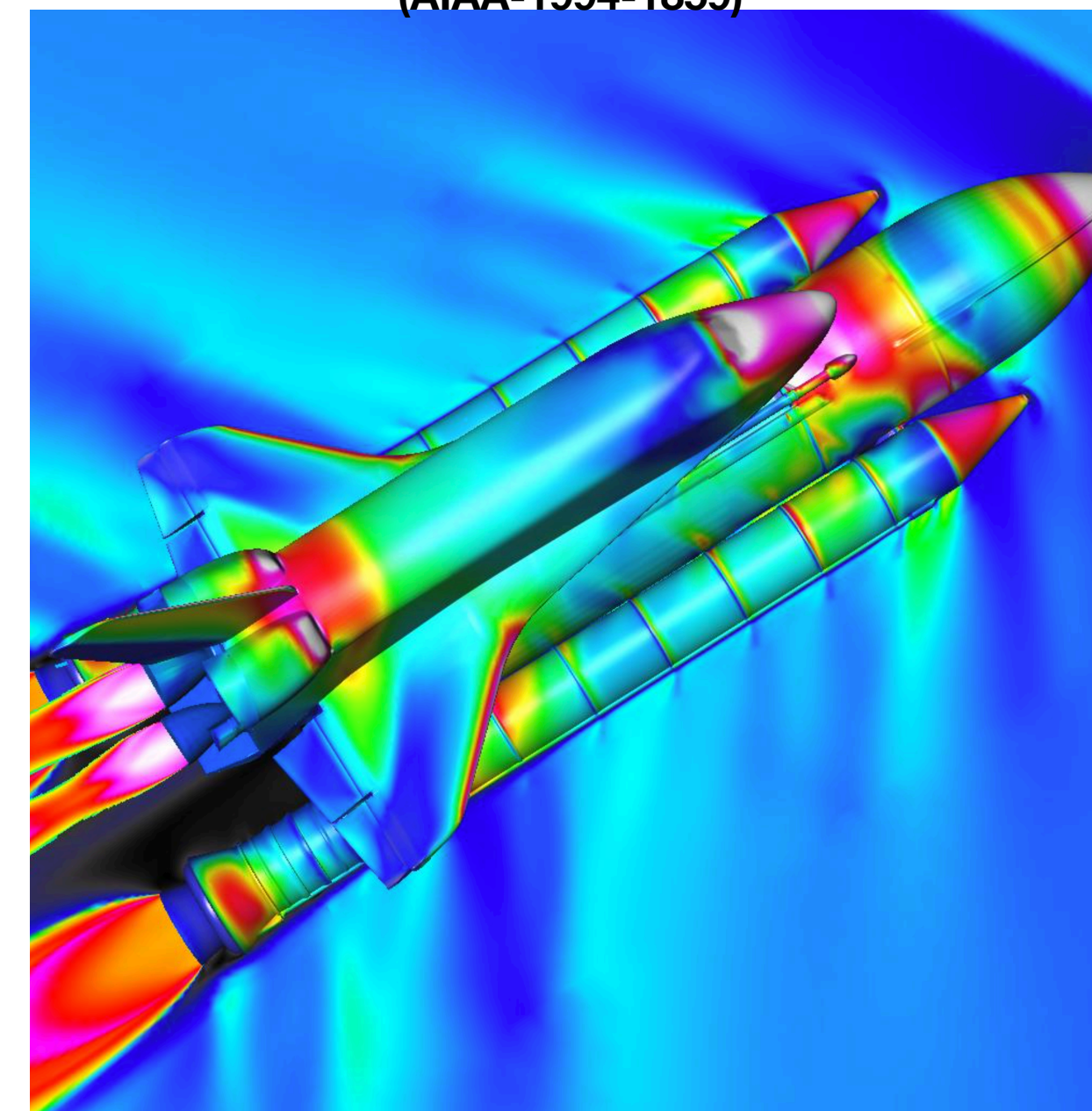
OVERFLOW

- Computational Fluid Dynamics (CFD) Software
- Structured, Finite-Difference Solver on Overset Grids
- Fortran 90 with no external dependencies
- CPU version has many years of hand tuning and is very performant (memory bound)
- Minimal experience with GPU programming before our 3 Hackathons

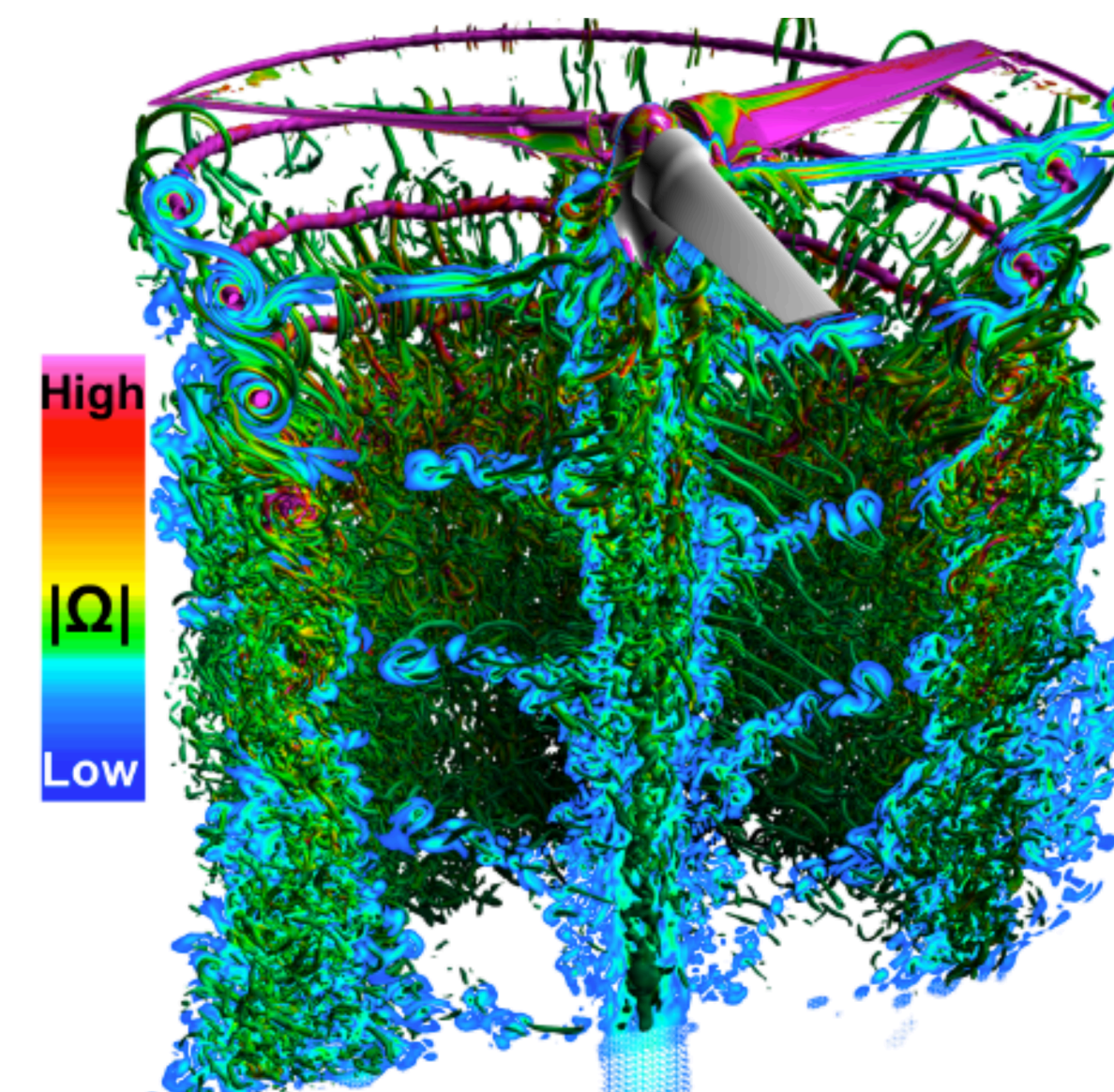
High-Lift Common Research Model Grid System
(AIAA-2017-0362)



Space Shuttle Launch Vehicle
Mach 1.25 Surface Pressure and Flow-Field Mach Number
(AIAA-1994-1859)



High Resolution Rotor Wake Simulation
(AHS 2011 Annual Forum)



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
- We created two miniapps for exploration at the hackathon

Miniapps

Central Solver Miniapp

- RHS and LHS calculation and solve (~60% total runtime)
- Stencil operations and batch scalar-pentadiagonal solve

Chimera Boundary Exchange Miniapp

- Overset boundary exchange (not a significant portion of total runtime)
- Interpolate, Pack, Transfer, Unpack

Learning to Fly - Our First Hackathon

In person - ORNL 2019

How to port to GPUs (at least our approach)

1. Add `!$ACC PARALLEL LOOP` pragmas to loops of interest
2. Create data regions
 - Replace implicit `COPYIN` and `COPYOUT` with explicit movement
3. Profile application to determine how to improve performance
 - Expose more parallelism
 - Do more work in a kernel
 - Hide launch latency with `ASYNC(stream)`

2nd Hackathon

Virtual - NAS 2020

- Wanted to learn how to leverage multiple GPUs
- 1 MPI Rank per GPU
- Were able to use CUDA-aware MPI to easily and efficiently move data between the GPUs

```
!$ACC HOST_DATA USE_DEVICE(buff) IF_PRESENT  
    call MPI_Irecv( buff, count, my_mpi_real, source, tag,      &  
                   comm, recv_request(n_recvs), ierr )  
!$ACC END HOST_DATA
```

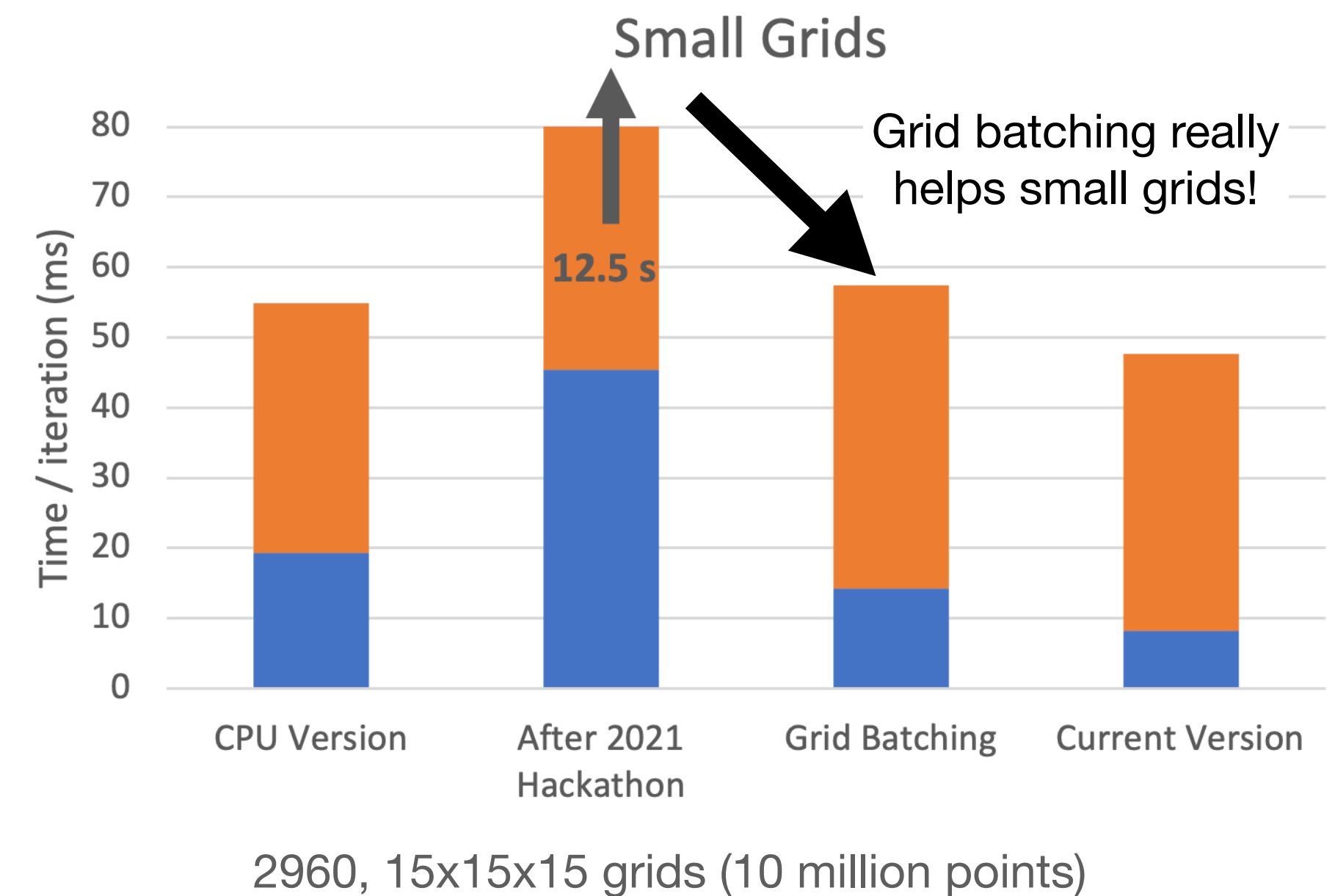
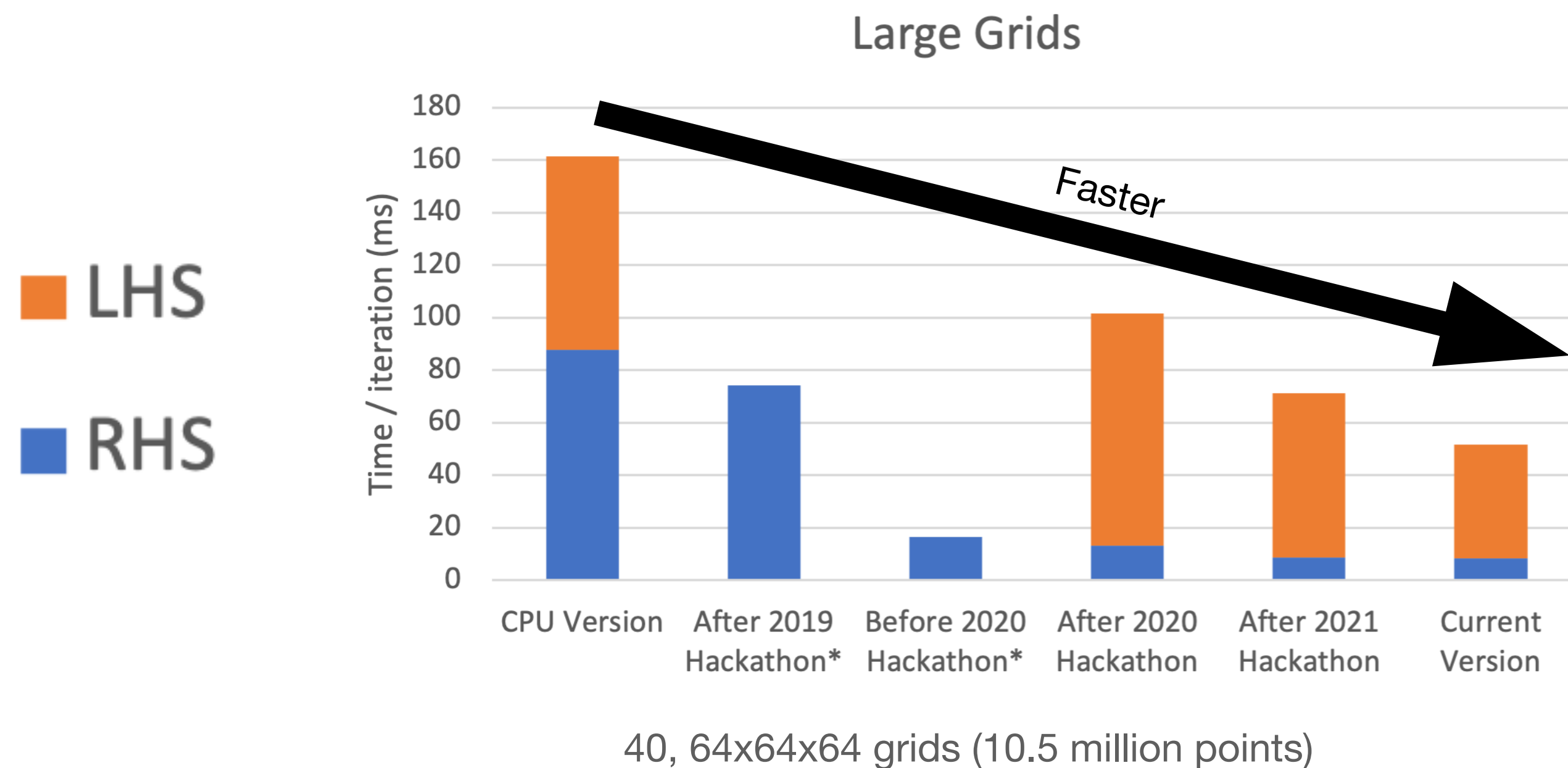

3rd Hackathon

Virtual - LaRC 2021

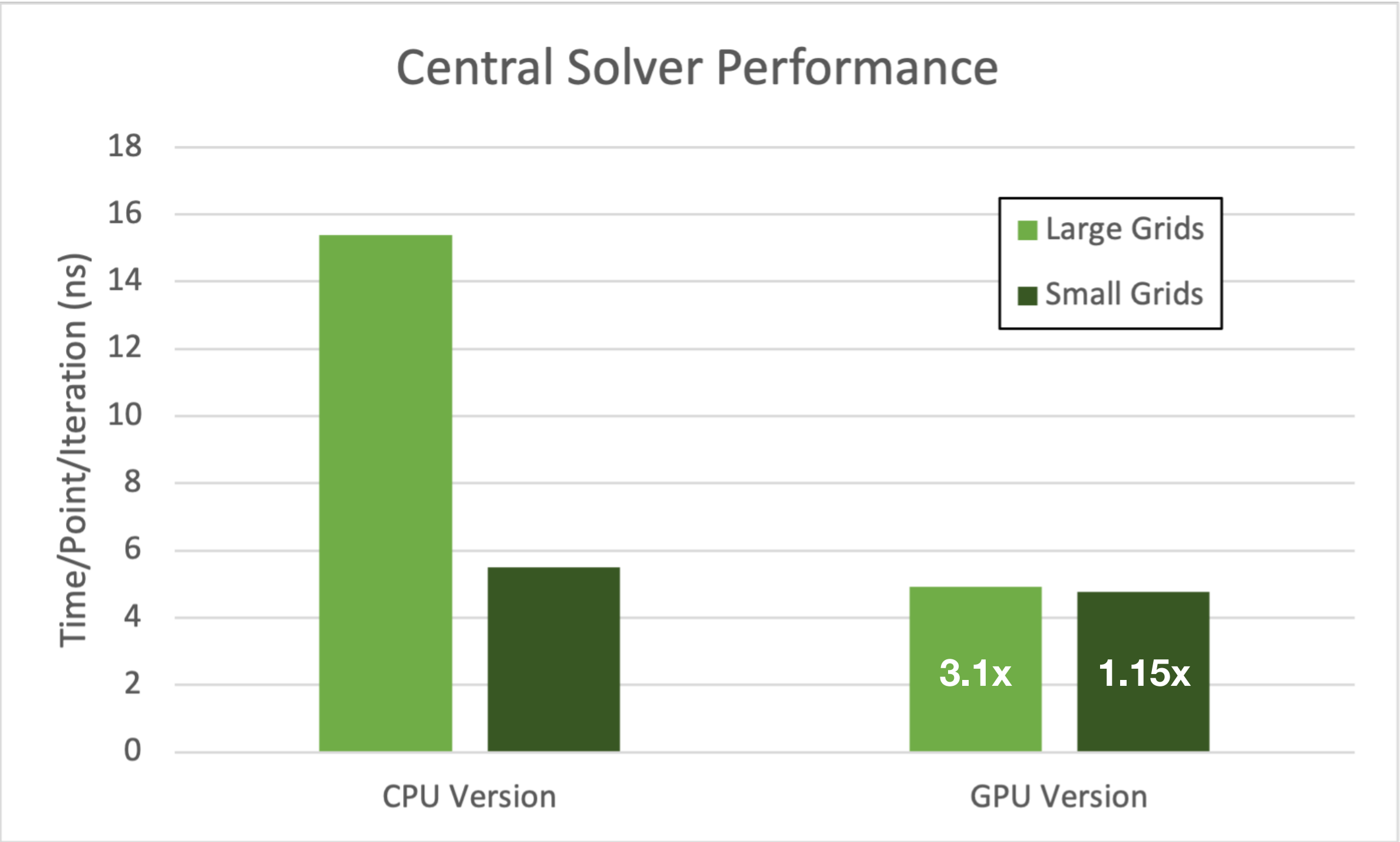
- Wanted to learn how to use the profiling tools to really get into the nitty-gritty
- Rewrote the linear solver with CUDA-Fortran
- Grouped several kernels together to provide more work per kernel and reduce the amount of memory being loaded
- Reorganized kernels to reduce register pressure
- Discussed batching the grids together for improving performance for lots of small grids

Hackathon Progress

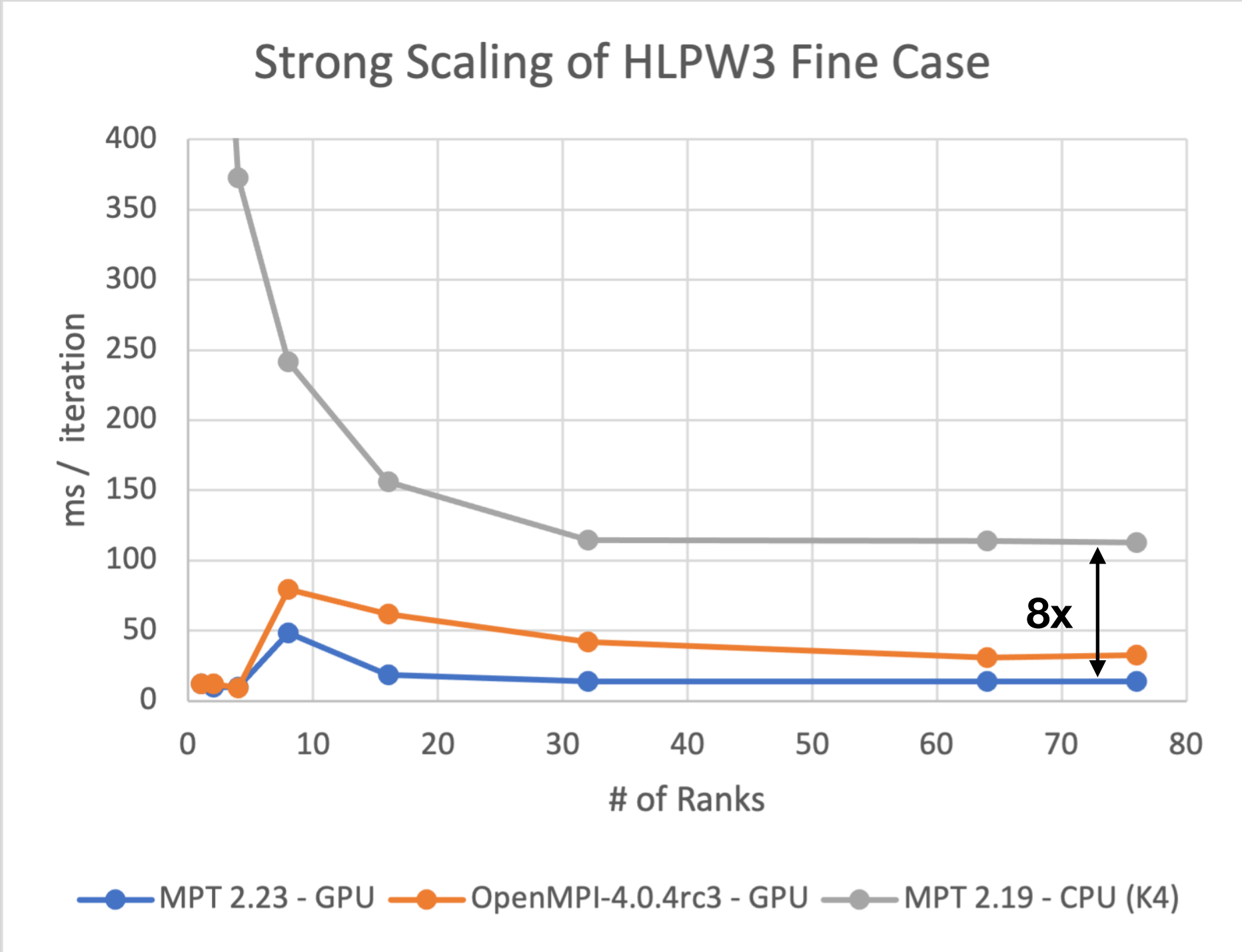
Continual improvement throughout our hackathon experiences



OVERFLOW Miniapps, You're Flying!



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



V100 GPU at NAS vs.
Dual Socket Intel Gold 6148 Skylake CPU at LaRC

To Infinity And Beyond

Moving Mini-apps into Full Application

- Not single source since large organizational and algorithmic changes for the most part
- Branch off after inputs for the GPU path
 1. Ensure the requested options have been ported
 2. Transfer the necessary data
 3. Run the solver on the GPU

Takeaways

Hackathons

- Great ways to get started and learn the basics of GPU programming
- Developed great contacts over the years of people in industry, government, and academia
 - People to discuss ideas with and learn from
 - Contacts with vendor and facility staff to address issues and collaborate
 - In the same “room” with world experts who are there to help you!

Takeaways

Hackathons

- Great to have the time set aside to work only on the GPU port
 - Minimal distractions with other work
 - Great to be able to have dedicated time for pairing and working with others
- I think the in-person hackathon was the most beneficial
 - Virtual was harder to block off time for multiple people (lots of distractions)
 - Less collaboration (lunch, breaks, dinners, being able to just walk over to another expert/group, etc)

Takeaways

OpenACC

- Quick and easy way to get your code on GPUs (about two days to have most of the miniapp running on GPUs)
- Allows us to get most of the performance out of the miniapp
- We do use CUDA-Fortran for certain kernels to get a little more flexibility specifically with nested gang / vector parallelism
- We have run into several compiler issues that have been reported (some fixed). Only having `nvfortran` isn't ideal.
- OpenMP seems to be the only approach that will be supported for Fortran offloading across different vendors

Acknowledgements

- Would like to thank NASA's Revolutionary Vertical Lift Technology project for funding this work.
- Would like to thank all those who help organize the Hackathons we attended and the experts we worked with.
- Would like to thank NVIDIA's Jeff Larkin and David Appelhans for their extensive work with us outside of the organized Hackathons.

Questions?

Backup Slides

Batch Operations

```
!$acc parallel loop gang collapse(2) async      &
!$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
end do
```


Flexibility in Gang Parallelism

Problem:

For non-tightly nested loops we cannot easily collapse or tile

```
do ig = 1, ngrids
  do l = 1, ld(ig)
    do k = 1, kd(ig)
      do j = 1, jd(ig)
        ! Do something here
      end do
    end do
  end do
end do
```

CUDA solution:

Apply multiple blocks to the grid and L loops

```
! Blocks across ig and l
do ig = blockIdx.y, ngrids, gridDim.y
  do l = blockIdx.x, ld(ig), gridDim.x
    ! Threads across j and k
    do k = threadIdx.y, kd(ig), blockDim.y
      do j = threadIdx.x, jd(ig), blockDim.x
        ! Do something here
      end do
    end do
  end do
end do
```

Flexibility in Gang Parallelism

Current OpenACC solution:

Clunky rewrite to collapse grid and L loops

```
!$acc parallel loop gang collapse(2) async &  
!$acc      present(jd,kd,ld)  
do ig = 1, ngrids  
  do l = 1, lmax  
    if( l <= ld(ig) ) then  
      !$acc loop vector collapse(2)  
      do k = 1, kd(ig)  
        do j = 1, jd(ig)  
          ! Do something here  
        end do  
      end do  
    end if  
  end do  
end do
```

Flexibility in Gang Parallelism

Potential OpenACC solution:

Be able to have multiple loops marked with gang parallelism

```
!$acc parallel loop gang(dim:2) async &  
!$acc    present(jd,kd,ld)  
do ig = 1, ngrids  
!$acc parallel loop gang  
  do l = 1, ld(ig)  
    !$acc loop vector collapse(2)  
    do k = 1, kd(ig)  
      do j = 1, jd(ig)  
        ! Do something here  
      end do  
    end do  
  end do  
end do
```