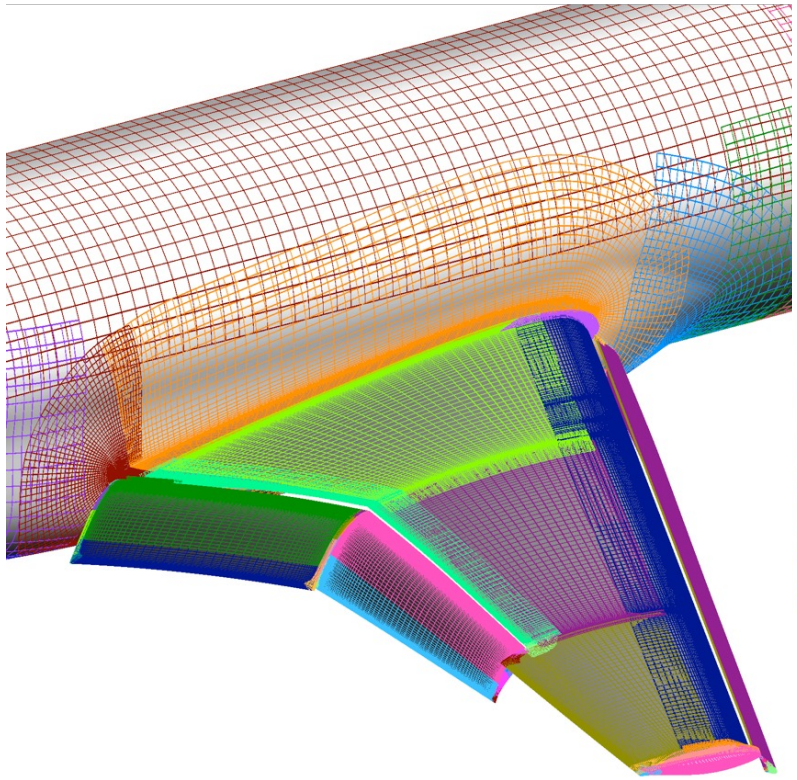
# Strategies for the GPU Implementation of the OVERFLOW CFD Code


| | |
|---|---|
| Chip Jackson | NASA Langley |
| David Appelhans | NVIDIA |
| Joe Derlaga | NASA Langley |
| Pieter Buning | NASA Langley |

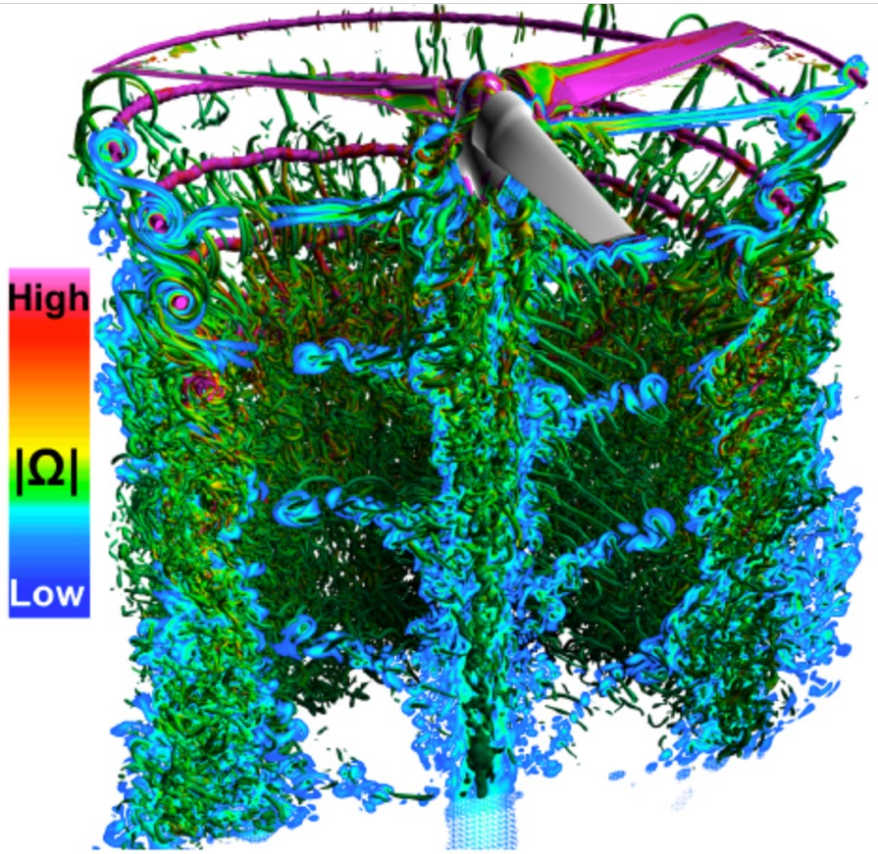

2024 GTC

# OVERFLOW

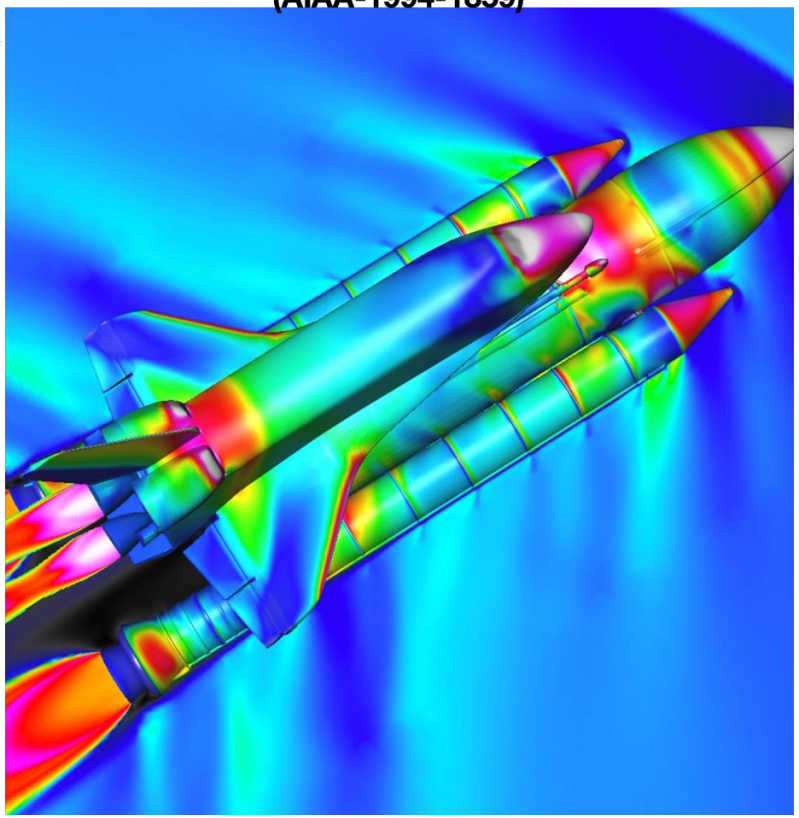
**High-Lift Common Research Model Grid System**
**(AIAA-2017-0362)**



**High Resolution Rotor Wake Simulation**
**(AHS 2011 Annual Forum)**



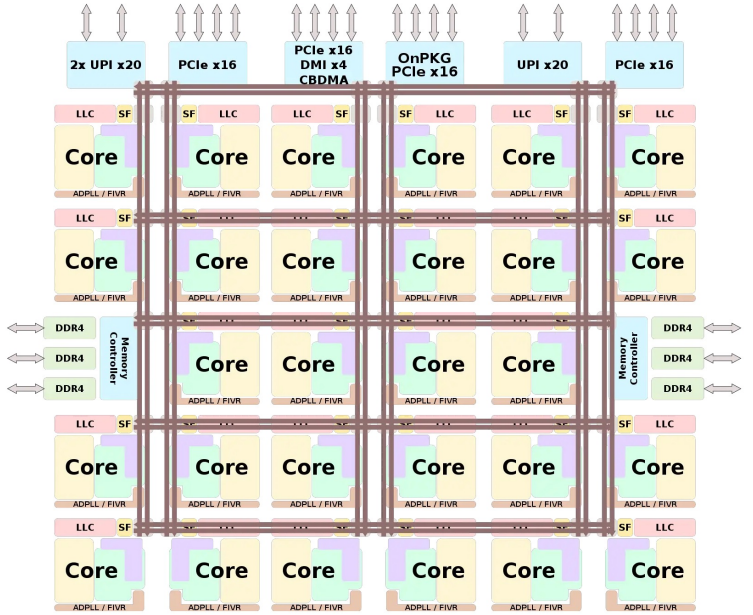

**Space Shuttle Launch Vehicle**
**Mach 1.25 Surface Pressure and Flow-Field Mach Number**
**(AIAA-1994-1859)**



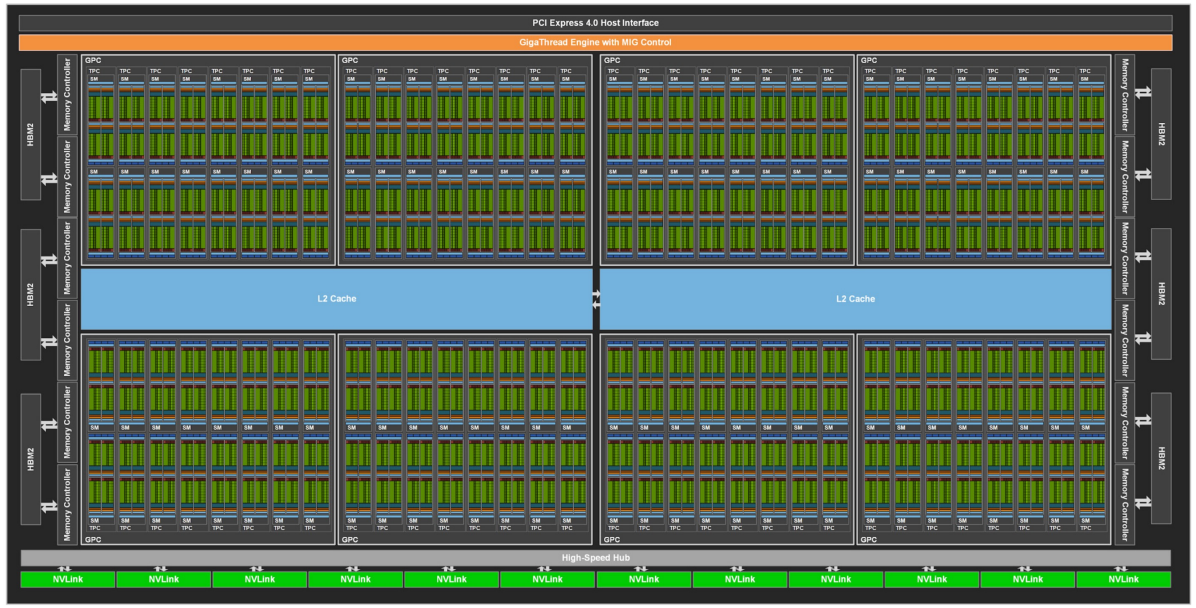
# Why GPUs?

- Higher Performance
  - Memory Bandwidth
  - Parallelism

- FLOPs/W
- FLOPs/$
- FLOPs/sqft




https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)




NVIDIA A100 Tensor Core GPU Architecture Whitepaper

# How do I run on GPUs?

- Many Options
- We selected OpenACC, CUDA Fortran, and CUDA C++

# Can I just add comments?

Yes, but…

# Expose more parallelism!

# Loop Grouping

```
subroutine time_step()
    do ig = 1, ngrids
        call do_everything_on_a_grid()
    end do
end subroutine time_step

subroutine do_everything_on_a_grid()
!$omp loop
    do l = 1, ld
        ! Work on an l-plane
        call sub1(l,...)
        call sub2(l,...)
        call sub3(l,...)
        ...
    end do
!$omp loop
    do k = 1, kd
        ! Work on a k-plane
        call sub1_k(k,...)
        ...
    end do
end subroutine do_everything_on_a_grid

subroutine sub1(l,...)
    do k = 1, kd
    do j = 1, jd ! Vectorized loop
        ! Do only one thing
    end do
    end do
end subroutine sub1
```

# Loop Grouping

```
subroutine time_step()
    do ig = 1, ngrids
        call do_everything_on_a_grid()
    end do
end subroutine time_step

subroutine do_everything_on_a_grid()
!$omp loop
    do l = 1, ld
        ! Work on an l-plane
        call sub1(l,...)
        call sub2(l,...)
        call sub3(l,...)
        ...
    end do
!$omp loop
    do k = 1, kd
        ! Work on a k-plane
        call sub1_k(k,...)
        ...
    end do
end subroutine do_everything_on_a_grid

subroutine sub1(l,...)
    do k = 1, kd
    do j = 1, jd ! Vectorized loop
        ! Do only one thing
    end do
    end do
end subroutine sub1
```

```
subroutine time_step()
    do ig = 1, ngrids
        call do_everything_on_a_grid()
    end do
end subroutine time_step

subroutine do_everything_on_a_grid()
    ! Work on a grid
    call sub1()
    call sub2()
    call sub3()
    ...
end subroutine do_everything_on_a_grid

subroutine sub1()
!$acc parallel loop collapse(3) async
    do l = 1, ld
    do k = 1, kd
    do j = 1, jd
        ! Do only one thing
    end do
    end do
    end do
end subroutine sub1
```

# What if your grids are very small?

# Grid Batching

```
subroutine time_step()
    do ib = 1, nbatch
        call do_everything_on_a_batch(batch(ib),...)
    end do
end subroutine time_step

subroutine do_everything_on_a_batch(batch,...)
    ! Work on a batch of grids
    call sub1(batch,...)
    call sub2(batch,...)
    call sub3(batch,...)
    ...
end subroutine do_everything_on_a_batch

subroutine sub1(batch,...)
!$acc parallel loop gang collapse(2) async    &
!$acc    present(batch,grid)                  &
!$acc    private(ig)
    do iig = 1, batch%ngrids
    do l   = 1, batch%lmax
        ! Gang parallelism over grids in a batch and l-planes
        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
                ! Vector parallelism within an l-plane
                ! Do only one thing
            end do
            end do

        end if
    end do
    end do
end subroutine sub1
```

# Code Modifications

1. **Expose Parallelism**
2. **Reduce Memory Traffic**
3. **Improve Data Access**

- Data Movement
- Asynchronous Kernel Launching
- Loop Grouping and Grid Batching
- Data Ordering
- Merging Kernels
- Shared Memory
- Multiple GPUs

# Why do you need CUDA?

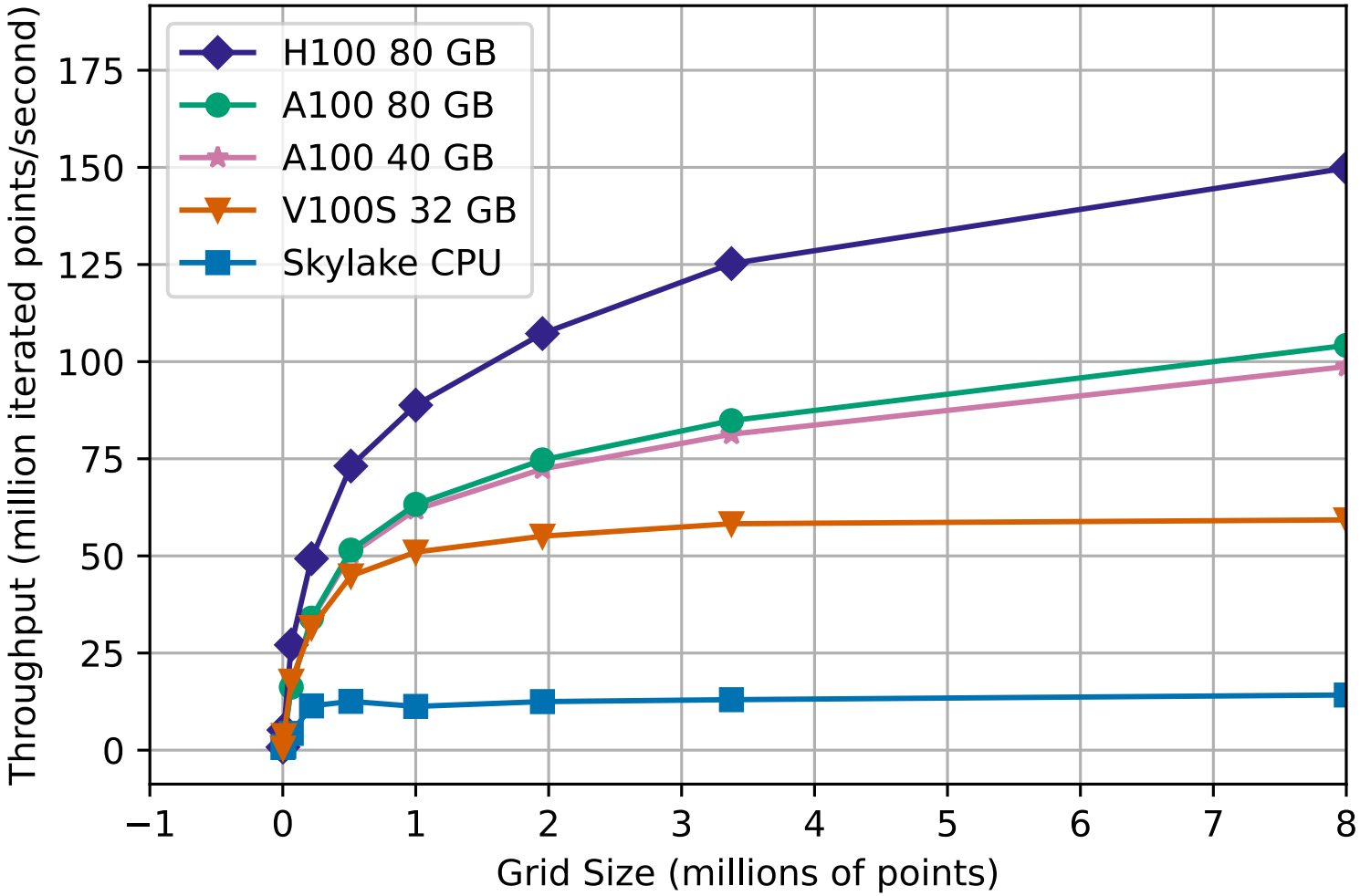
- CUDA Fortran:
  - Can more easily use shared memory
    - Explicit thread synchronization
  - Can express more complicated loop parallelization
    - For instance: OpenACC cannot have gang parallelism on multiple uncollapsed loops
- CUDA C++:
  - Templating can reduce register pressure, increasing occupancy and performance

# Current Status

- Significant changes resulted in two paths through OVERFLOW
  - Same startup, shutdown routines
  - Some shared routines, like boundary conditions and I/O routines
- Implemented a subset of the capabilities of the code with work ongoing
- Using GPU-aware MPI to communicate between multiple GPUs

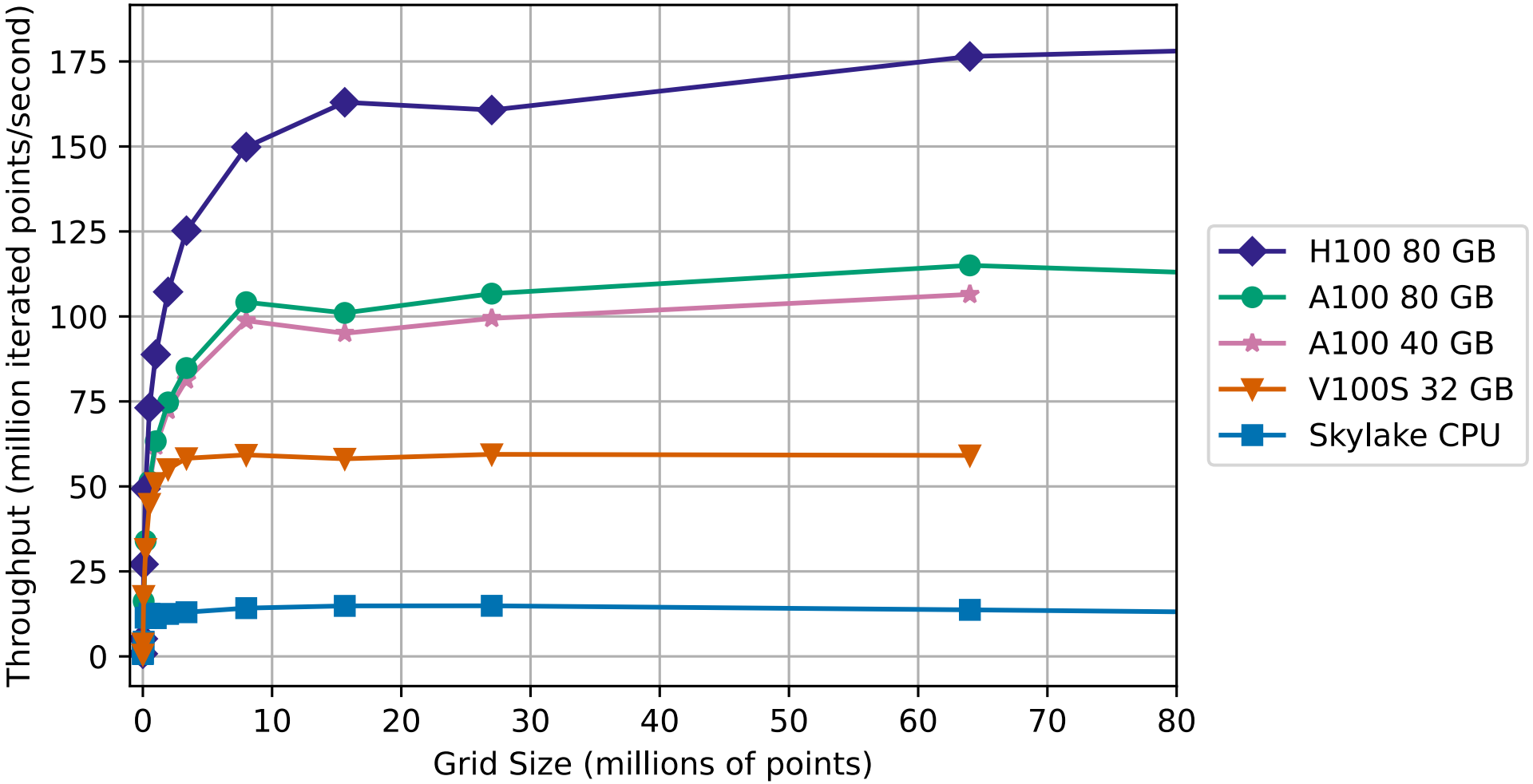
# How big of a grid do I run?

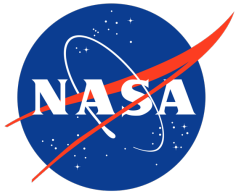
# How big of a grid do I run?




Skylake CPU is a dual socket, 20-core Xeon Gold 6148 node running at 2.4 GHz

# How big of a grid do I run?

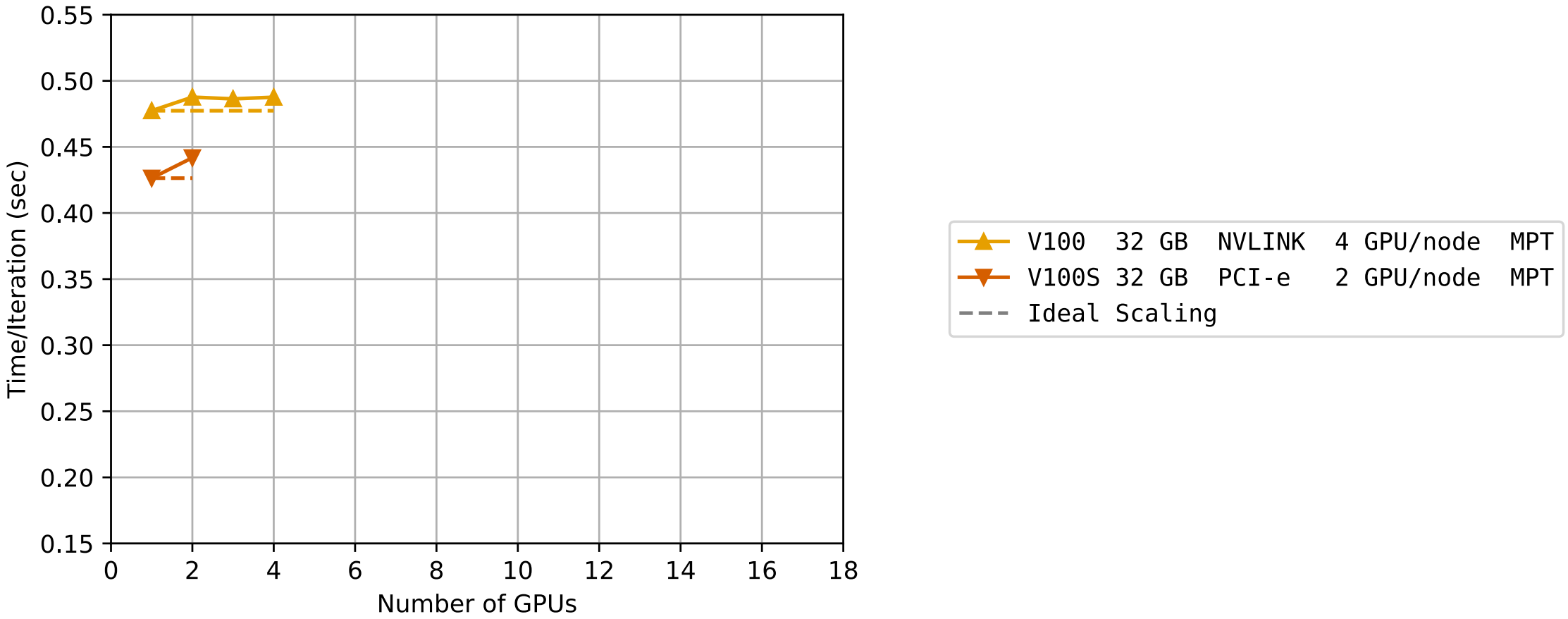




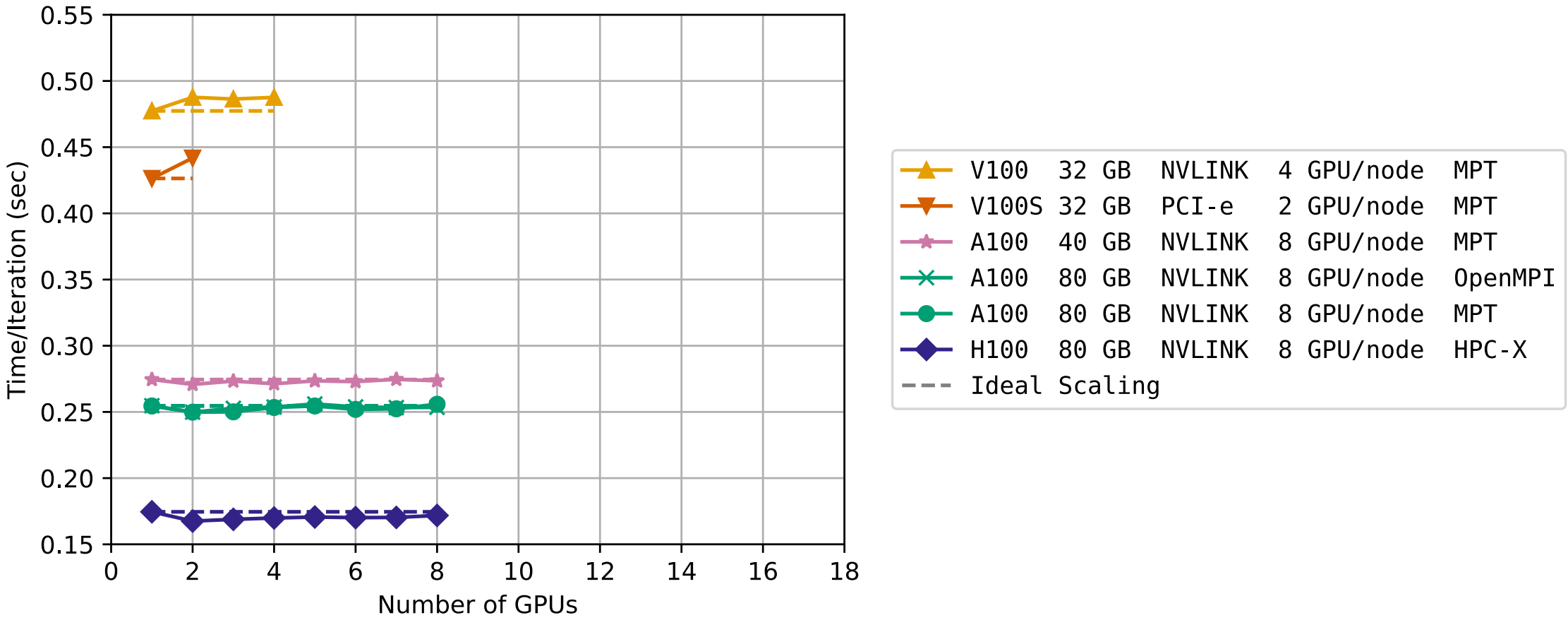

# How well does it run on multiple GPUs?

Weak Scaling

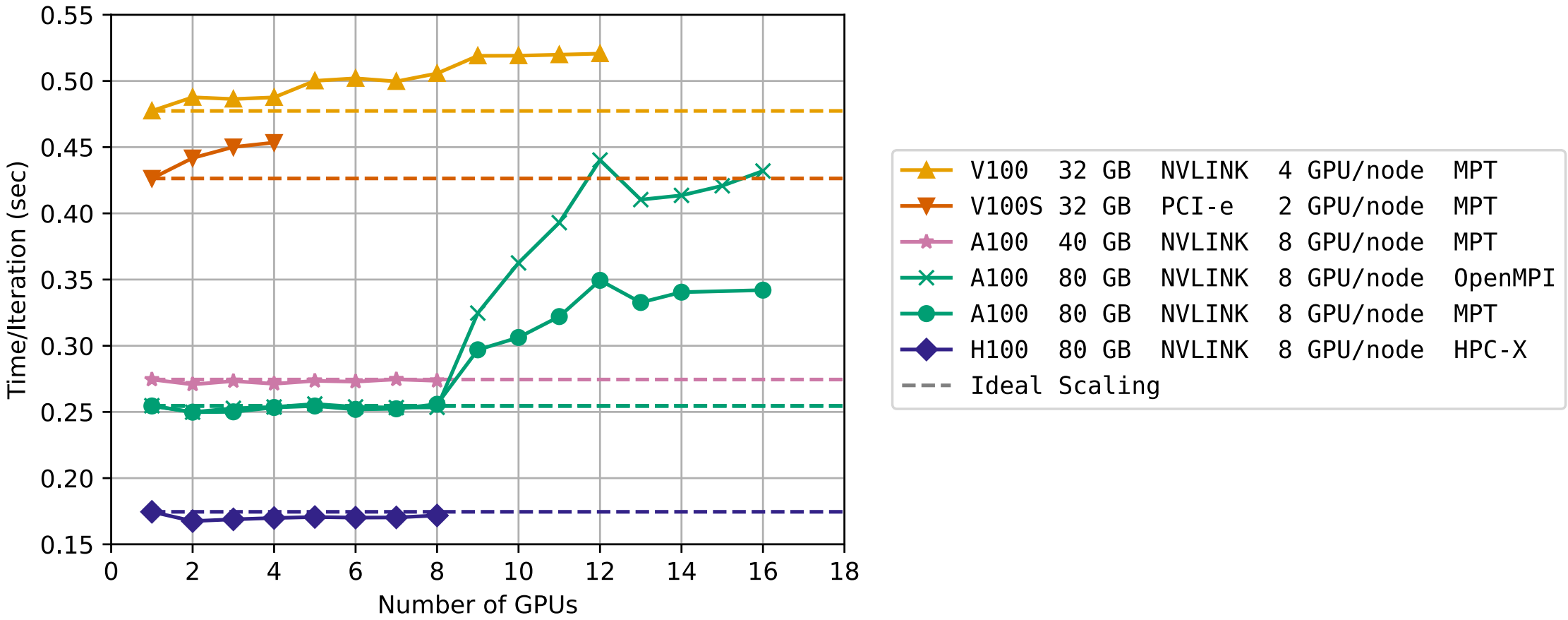
# How well does it run on multiple GPUs?



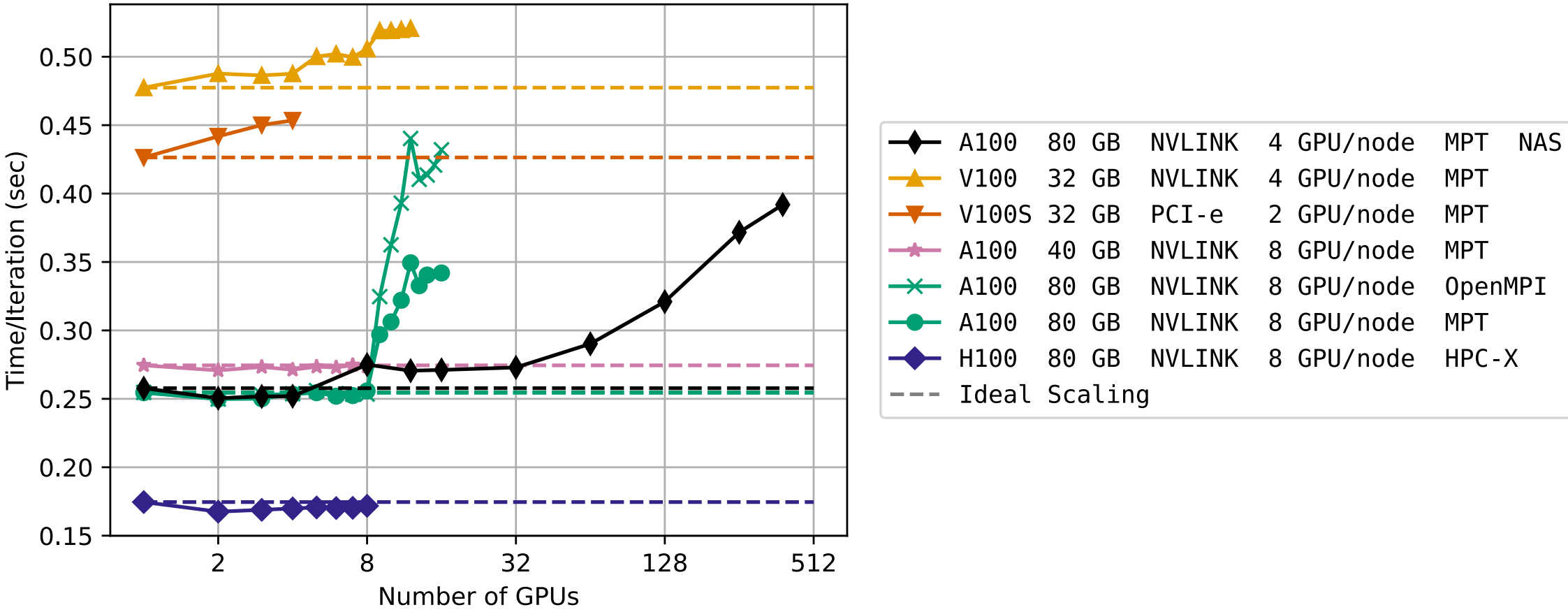
# How well does it run on multiple GPUs?



# How well does it run on multiple GPUs?

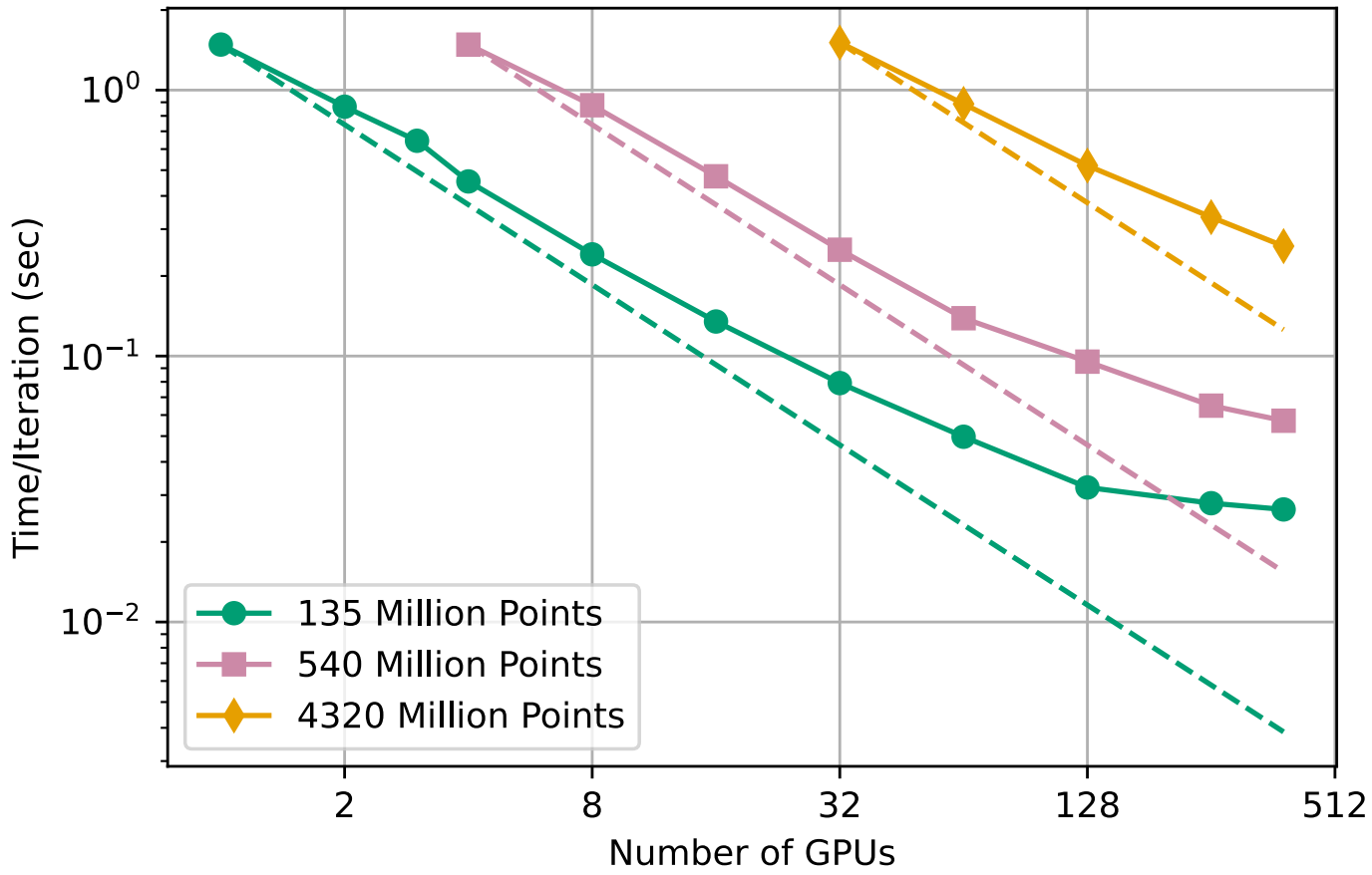


# How well does it run on multiple GPUs?



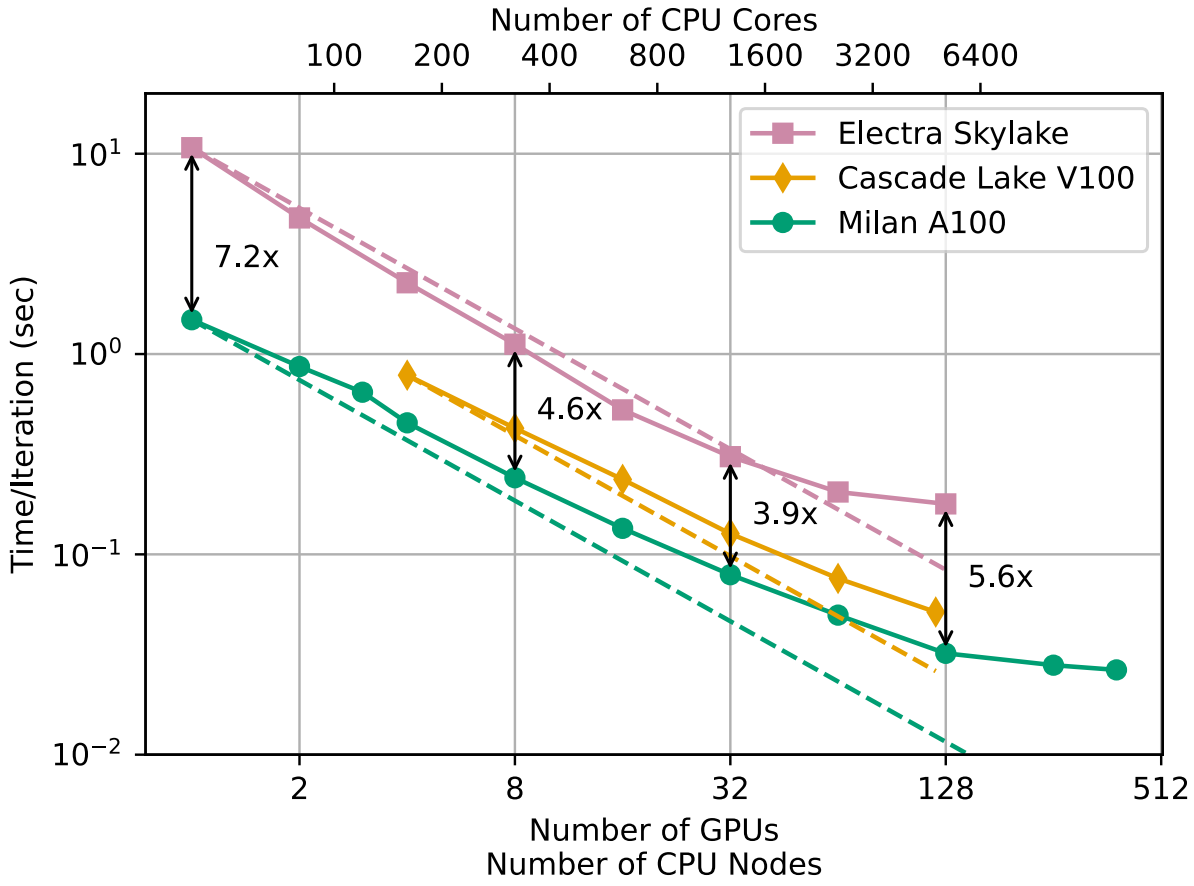
# What about strong scaling?

# What about strong scaling?




On new A100 Cabeus system at NAS (4 A100 80GB / node)

# How does strong scaling compare?

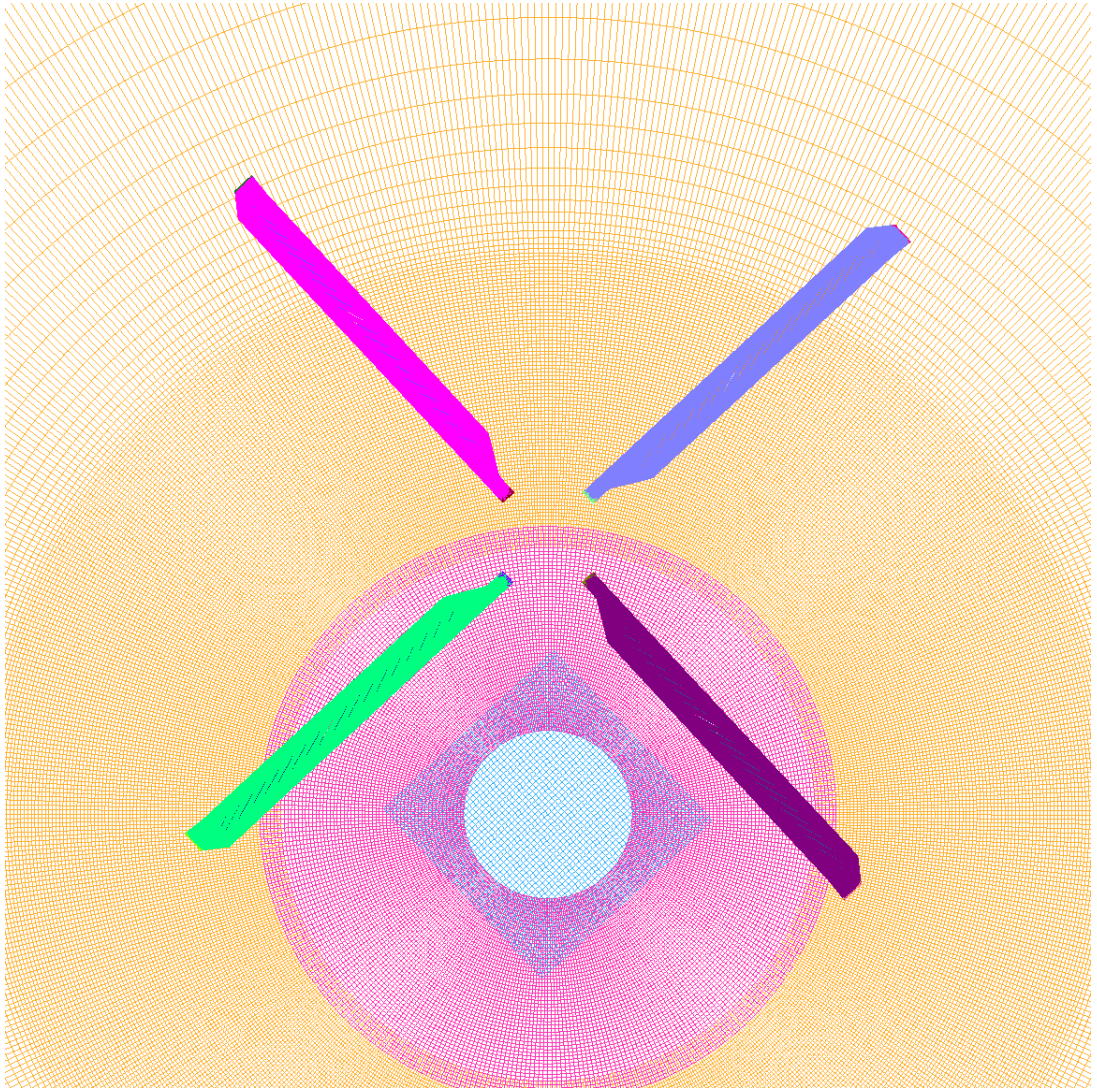



1 A100 is as fast as 270 Skylake Cores (7 nodes)
8 A100 is as fast as 2100 Skylake Cores (53 nodes)

135-million-point problem

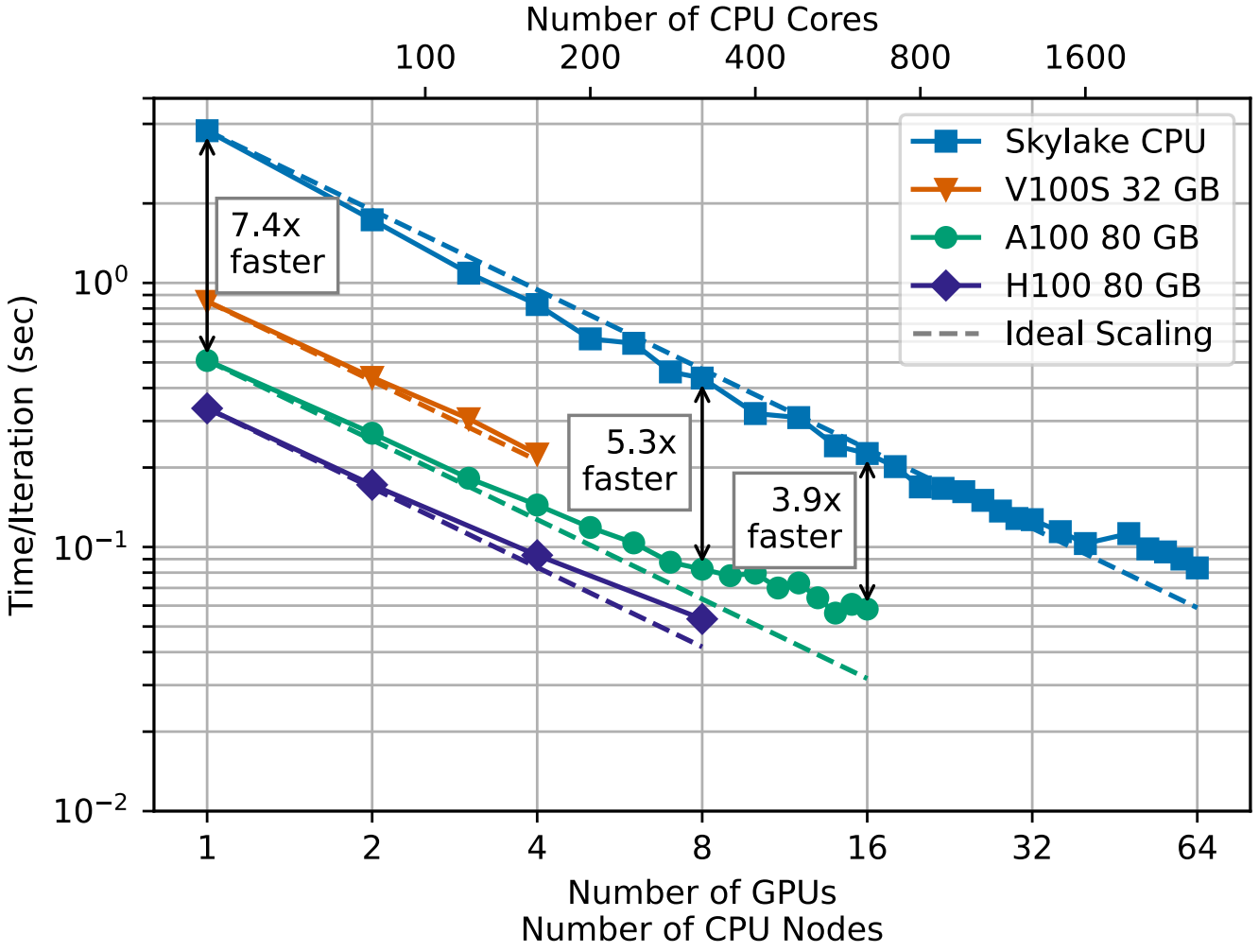
# What about a real problem?

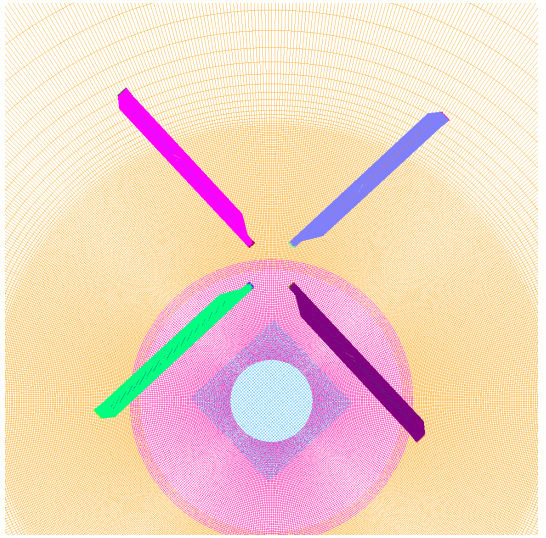
# Hover Validation Acoustic Baseline Case



15 grid blocks
53 million points

# HVAB Case



| GPU | Speedup Relative to Skylake Node | Equivalent Number of Skylake Cores |
|---|---|---|
| V100S | 4.4 | 156 |
| A100 | 7.4 | 265 |
| H100 | 11.3 | 390 |

# Concluding Remarks

- OVERFLOW is now able to run on GPUs
  - Central Difference Scheme, Scalar-Pentadiagonal LHS, Fixed and Moving Grids
  - Work is ongoing to add options (upwinding schemes, SSOR)
- Many lessons-learned and code modifications from this effort
- Running on GPUs is significantly faster than CPUs
  - 1 A100 GPU is 5 – 8 times faster than a dual socket Skylake node
  - 1 A100 GPU is equivalent to approximately 260 Skylake cores (7 nodes)
- Currently in testing mode and will be in the next major release of OVERFLOW

# More Information

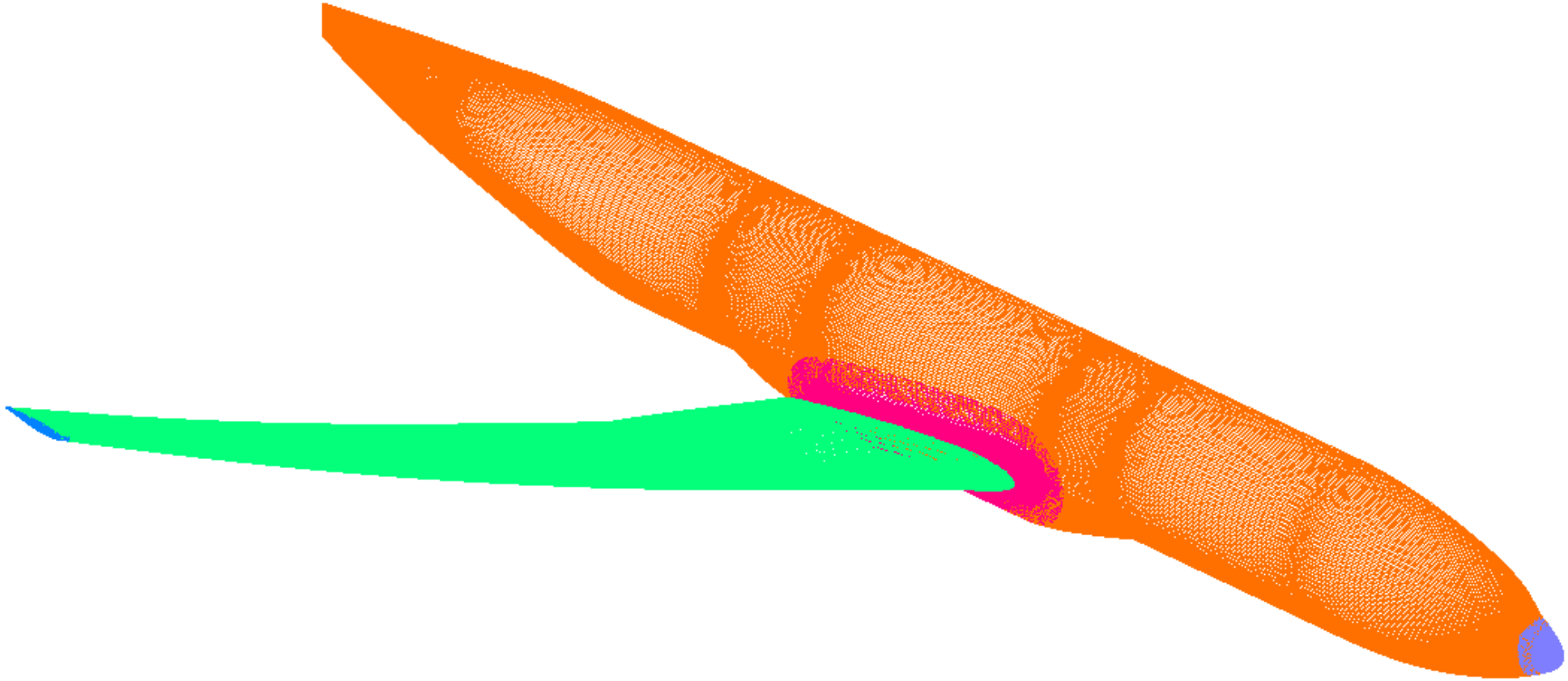
- For more details about the process of porting to GPUs, some code modifications, and our experience at OpenACC Hackathons see my talk “Porting OVERFLOW CFD Code to GPUs: To Hackathons and Beyond” https://youtu.be/ai0FNz4d70Q
- For more details about the code modifications and performance presented here see AIAA Paper 2024-0042 (available for free at https://ntrs.nasa.gov/citations/20230016176)

# Acknowledgements


- Would like to thank the NASA Revolutionary Vertical Lift Technology Project for funding this work.
- Also, thanks to NVIDIA and their technical staff for their support and assistance during this work.

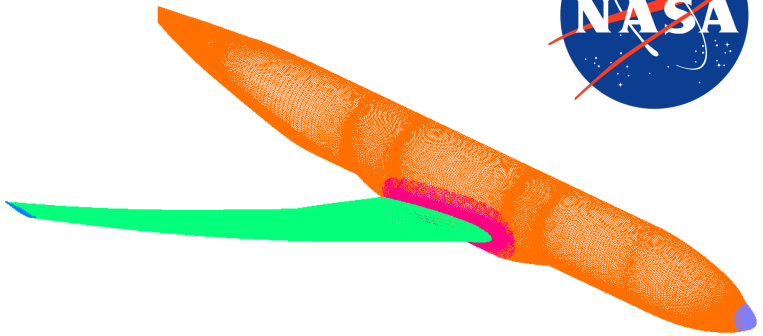
# Questions?

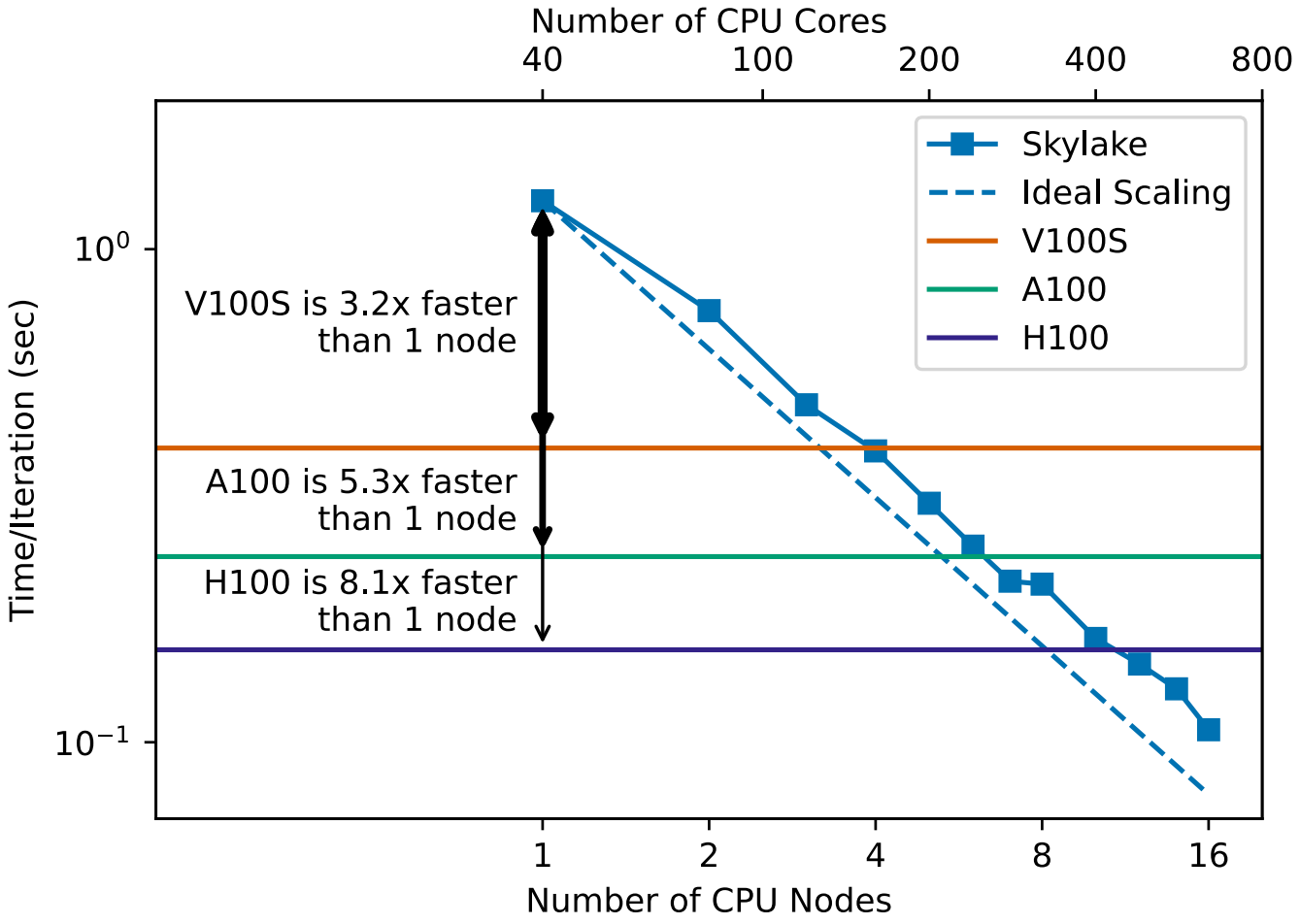
# Drag Prediction Workshop 6 Case



8 grid blocks
25 million points

# DPW6 Case






| GPU | Speedup Relative to Skylake Node | Equivalent Number of Skylake Cores |
|---|---|---|
| V100S | 3.2 | 158 |
| A100 | 5.3 | 253 |
| H100 | 8.1 | 437 |

# How do I run on GPUs?

| GPU Vendor | CUDA | | HIP | | SYCL | | Standard | | OpenMP | | OpenACC | | Kokkos | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | C | F | C | F | C | F | C | F | C | F | C | F | C | F |
| **NVIDIA** | ✓ | ✓ | ▼ | ⊘ | ▲ | ⊘ | ✓ | ✓ | ◆ | ◆ | ✓ | ✓ | ▲ | ⊘ |
| **AMD** | ▼ | ⊘ | ✓ | ⊘ | ▼ | ⊘ | ▲ | ⊘ | ✓ | ✓ | ▲ | ▲ | ▲ | ⊘ |
| **Intel** | ▼ | ⊘ | ⊘ | ⊘ | ✓ | ⊘ | ⊘ | ◆ | ✓ | ✓ | ⊘ | ⊘ | ▲ | ⊘ |

✓ Full Vendor Support

◆ Partial Vendor Support

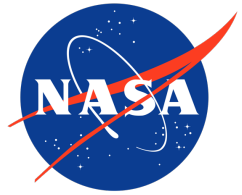
▼ Indirect Vendor Support

▲ Comprehensive support, not from vendor

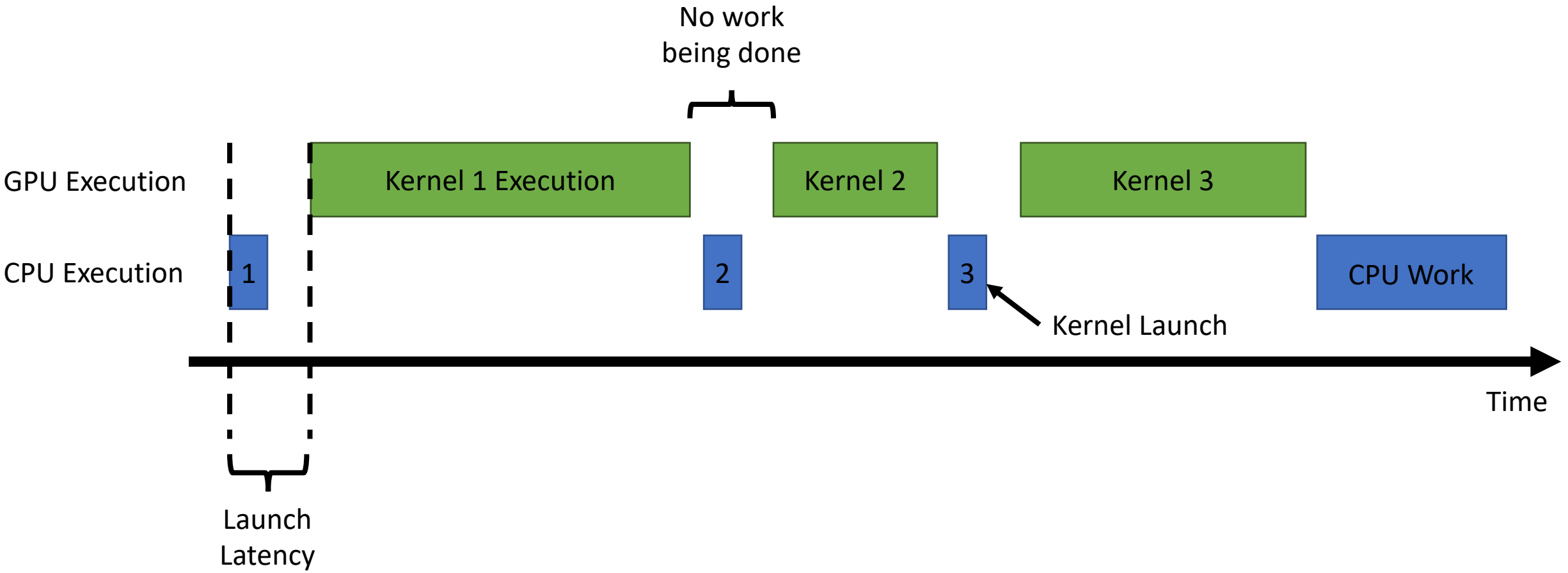
⊘ Very limited or no support

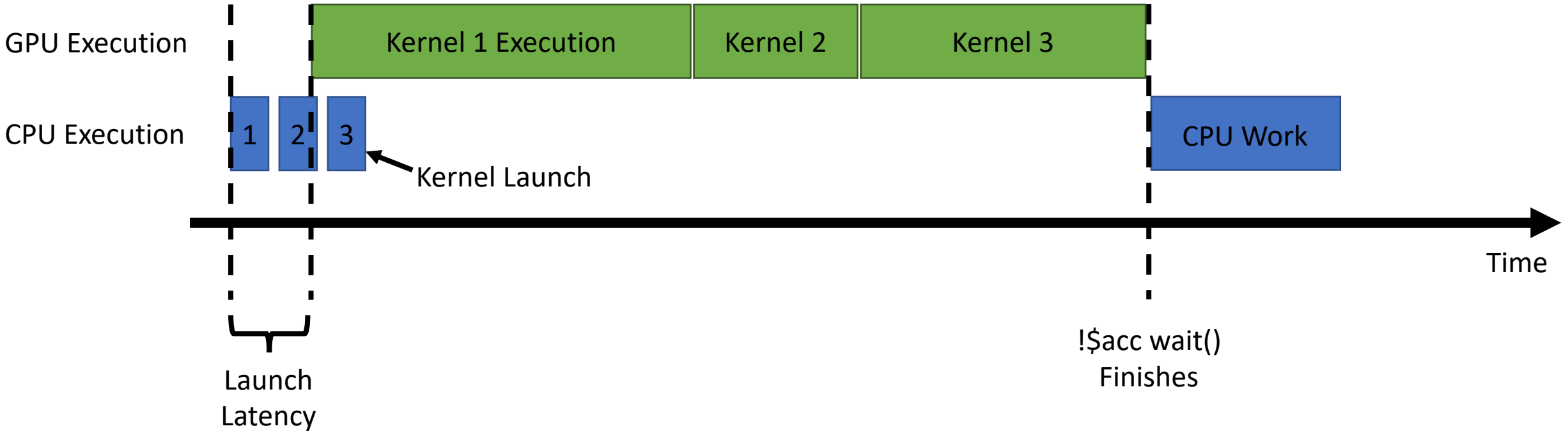
C = C++ (sometimes C), F = Fortran

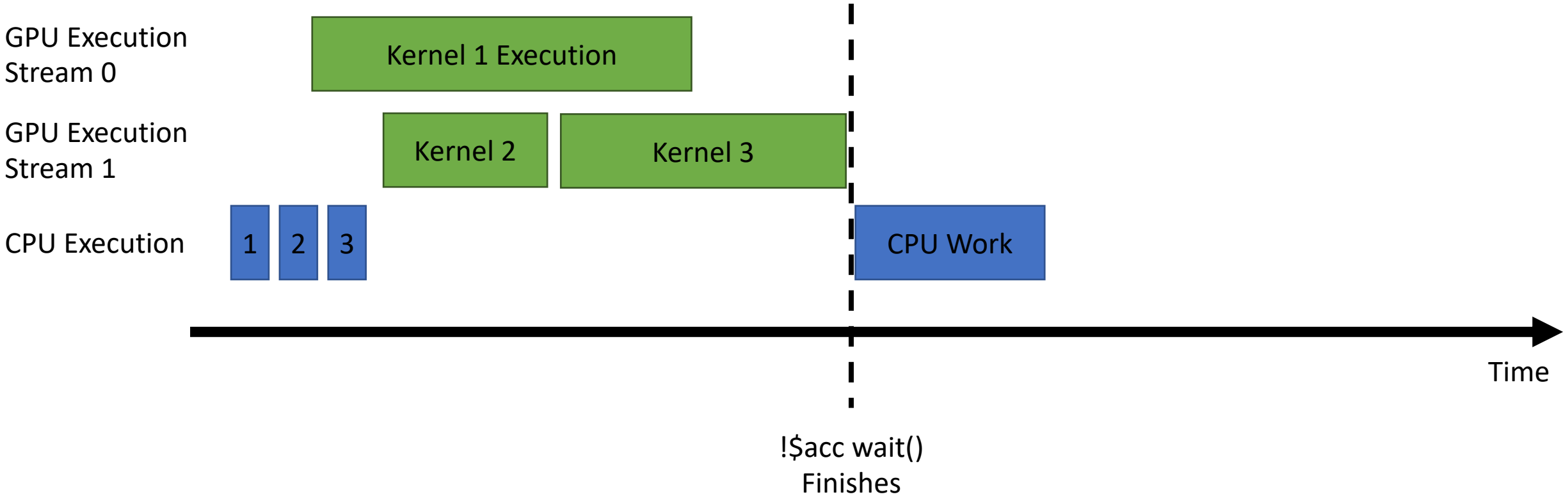
* As of November 2022 (adapted from Herten 2022)




# Get the CPU out of the way!


No work
being done
GPU Execution
Kernel 1 Execution
Kernel 2
Kernel 3
CPU Execution
1
2
3
CPU Work
Kernel Launch
Time
Launch
Latency


GPU Execution
Kernel 1 Execution
Kernel 2
Kernel 3
CPU Execution
1
2
3
Kernel Launch
CPU Work
Time
Launch
Latency
!$acc wait()
Finishes


GPU Execution Stream 0
Kernel 1 Execution
GPU Execution Stream 1
Kernel 2
Kernel 3
CPU Execution
1
2
3
CPU Work
Time
!$acc wait() Finishes