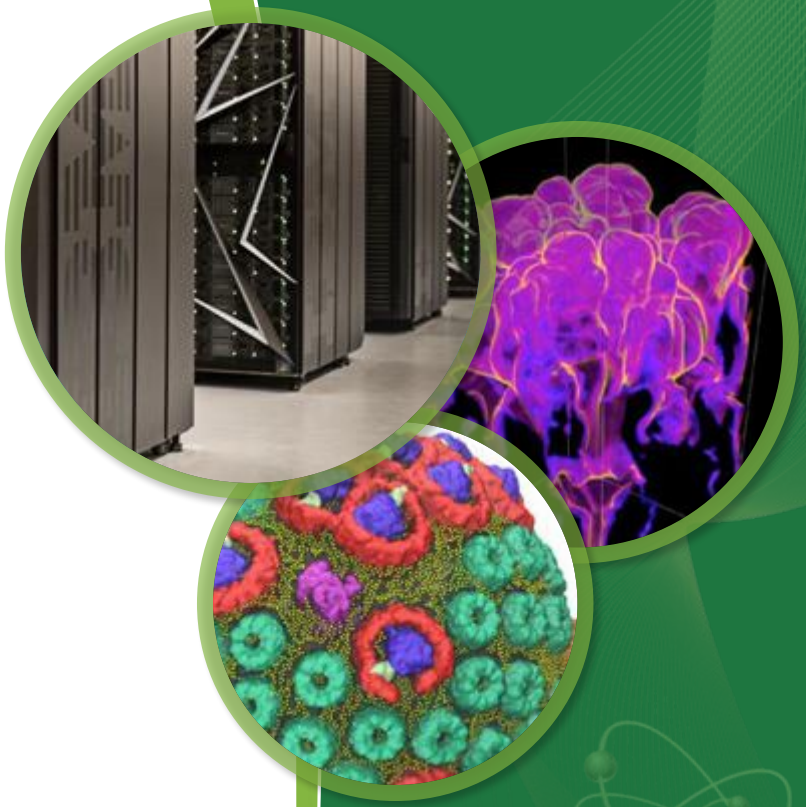
Targeting GPUs using OpenMP Directives on Summit with GenASiS: A Simple and Effective Fortran Experience



Reuben D. Budiardja
Scientific Computing Group,
Oak Ridge Leadership Computing Facility,
Oak Ridge National Laboratory

Christian Cardall
Physics Division,
Oak Ridge National Laboratory

ORNL is managed by UT-Battelle
for the US Department of Energy

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

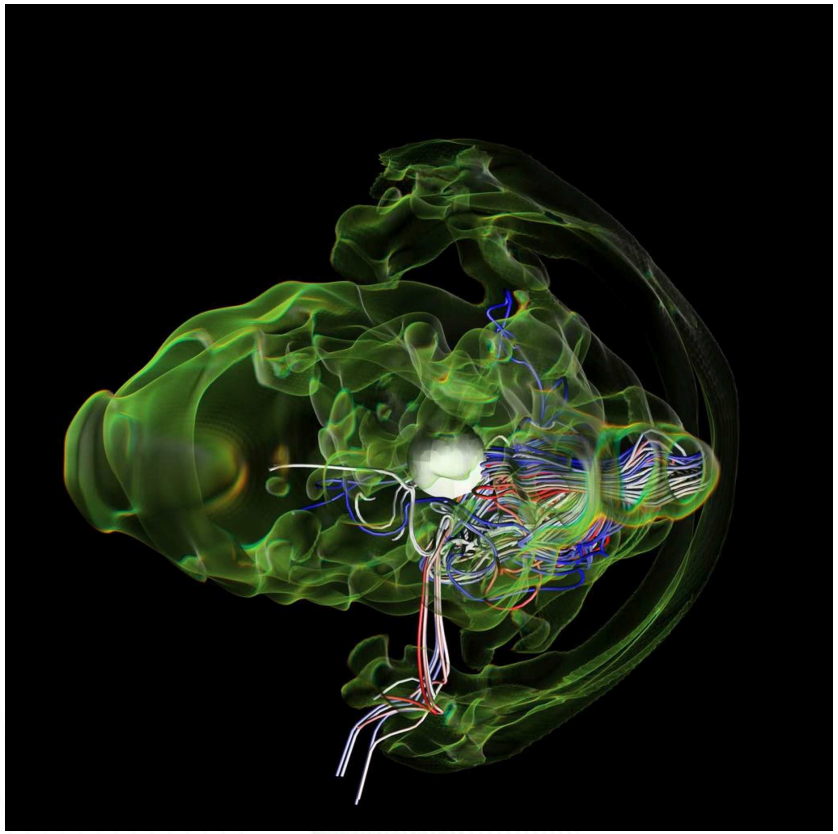
The Application

General ***A***strophysics ***Si***mulation ***S***ystem (GenASiS)

- Designed for parallel, large scale simulations
  - weak-scale to ~100 thousands MPI processes
- Written entirely in modern Fortran (2003, 2008)
- Modular, object-oriented design, and extensible
- Multi-physics solvers:
  - (Magneto)-hydrodynamics (HLL, HLLC solvers)
  - Explicit 2nd order time-integration
  - Self-gravity, polytropic & nuclear EoS
  - Grey and spectral neutrino transport
- CPU only code with OpenMP for threading (prior to this work)

The Application

- Studied the role fluid instabilities --- convection and Standing Accretion Shock Instability (SASI) --- in supernova dynamics
- Discovered exponential magnetic field amplification by SASI in progenitor star → origin of neutron star magnetic fields
- Refactored to three major subdivisions: *Basics*, *Mathematics*, *Physics* → allowing unit testing, ad-hoc/standalone tests, mini-apps



Paths to Targeting GPU

- CUDA
  - requires rewrite of all computational kernels
  - loss of Fortran semantics (multi-d arrays, pointer/array remapping)
  - requires interfacing with the rest of the (Fortran) code
- CUDA Fortran
  - non standard extension to Fortran (XL, PGI)
  - cannot easily fall back to standard Fortran
- Directives (OpenMP)
  - retain Fortran semantics
  - OpenMP 4.5 has excellent support from IBM XL (Summit), CCE (Titan)
    - with excellent support for modern Fortran

Lower-Level GenASiS Functionality

- Fortran wrappers to OpenMP APIs
  - `call AllocateDevice(Value, D_Value)`
    → `omp_target_alloc()`

    `call AssociateHost(D_Value, Value)`
    → `omp_target_associate_ptr()`

    `call UpdateDevice(Value, D_Value),`
    `call UpdateHost(Value, D_Value)`
    → `omp_target_memcpy()`

Value : Fortran array
D_Value : type(c_ptr), GPU address

- Affirmative control of data movement
- Persistent memory allocation on the device

Higher-level GenASiS Functionality

- StorageForm :
  - a class for data and metadata; the 'heart' of data storage facility in GenASiS
  - metadata includes units, variable names (for I/O, visualization)
  - used to group together a set of related physical variables (e.g. Fluid)
  - render more generic and simplified code for I/O, ghost exchange, prolongation & restriction (AMR mesh)

- `Data: StorageForm % Value ( nCells, nVariables )`

- Methods:
  - `call StorageForm % Initialize ( )` ← **allocate data on host**
  - `call StorageForm % AllocateDevice ( )` ← **allocate data on GPU**
  - `call StorageForm % Update{Device,Host} ( )` ← **transfer data**

Sidebar: OpenMP Memory for Offload

- OpenMP maps host (CPU) variables to device (GPU) (explicitly or implicitly)
  - default copy to-from
- *Presence check*: if fail, new variable is created on device
  - sometime requires explicit association to avoid unintentional data movement

Offloading Computational Kernel

```
 1 subroutine AddKernel ( A, B, D_A, D_B, D_C, C )
 2
 3   real ( KDR ), dimension ( : ), intent ( in ) :: A, B
 4   type ( c_ptr ), intent ( in ) :: D_A, D_B, D_C
 5   real ( KDR ), dimension ( : ), intent ( out ) :: C
 6
 7   integer ( KDI ) :: i
 8
 9   call AssociateHost ( D_A, A )
10   call AssociateHost ( D_B, B )
11   call AssociateHost ( D_C, C )
12
13   !$OMP target teams distribute parallel do schedule ( static, 1 )
14   do i = 1, size ( C )
15     C ( i ) = A ( i ) + B ( i )
16   end do
17   !$OMP end target teams distribute parallel do
18
19   call DisassociateHost ( C )
20   call DisassociateHost ( B )
21   call DisassociateHost ( A )
22
23 end subroutine AddKernel
```

Tells OpenMP data location on GPU
→avoid (implicit) allocation & transfer

```
call F % Initialize &
                    iables])
                    eDevice ( )
call F % UpdateDevice ( )
call AddKernel &
  ( F % Value ( :, 1 ),
    F % Value ( :, 2 ), &
    F % D_Value ( 1 ),
    F % D_Value ( 2 ), &
    F % D_Value ( 3 ),
    F % Value ( :, 3 ) )
```

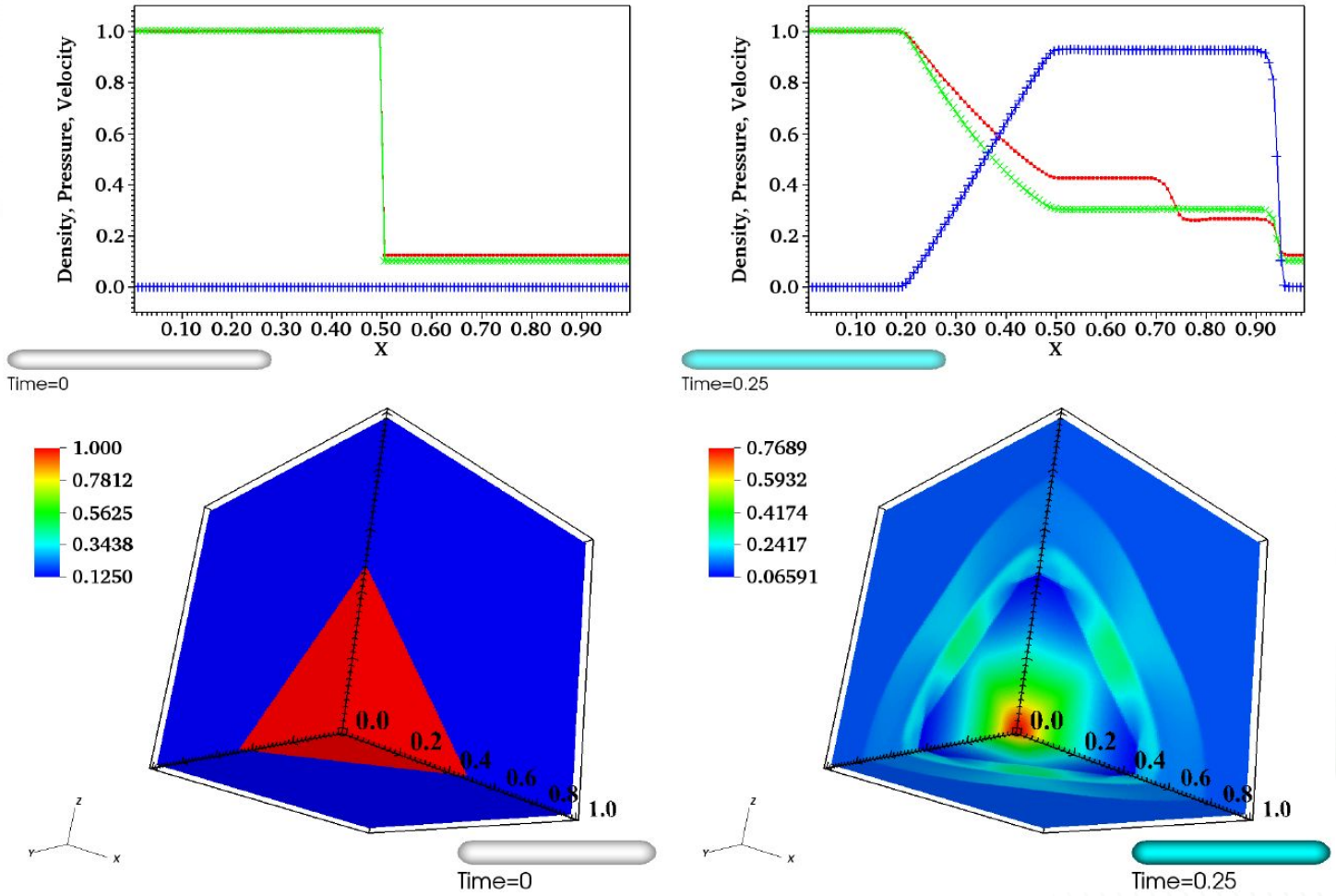
Example of Kernel with Pointer Remapping

```
real ( KDR ), dimension ( :, :, : ), pointer
  :: V, dV

V ( -1:nX+2, -1:nY+2, -1:nZ+2 ) => F % Value ( : , iV )
dV ( -1:nX+2 , -1:nY+2 , -1:nZ+2 ) => dF % Value ( : , iV )

call ComputeDifferences_X ( V, F % D_Value ( iV), ... )
```

Example of Kernel with Pointer Remapping

```
 1 subroutine ComputeDifference_X ( V, D_V, D_dV, dV )
 2
 3   real ( KDR ), dimension ( -1:, -1:, -1: ), intent ( in ) :: V
 4   type ( c_ptr ), intent ( in ) :: D_V, D_dV
 5   real ( KDR ), dimension ( -1:, -1:, -1: ), intent ( out ) :: dV
 6
 7   integer ( KDI ) :: i, j, k
 8
 9   call AssociateHost ( D_V, V )
10   call AssociateHost ( D_dV, dV )
11
12   !$OMP target teams distribute parallel do collapse ( 3 ) schedule ( static, 1 )
13   do k = 1, nZ
14     do j = 1, nY
15       do i = 0, nX + 2
16         dV ( i, j, k ) = V ( i, j, k ) - V ( i - 1, j, k )
17       end do
18     end do
19   end do
20   !$OMP end target teams distribute parallel do
21
22   call DisassociateHost ( dV )
23   call DisassociateHost ( V )
24
25 end subroutine ComputeDifferences_X
```

Porting a Fluid Dynamics Application: RiemannProblem



Initial (left) and final (right) density of 1D and 3D RiemannProblem

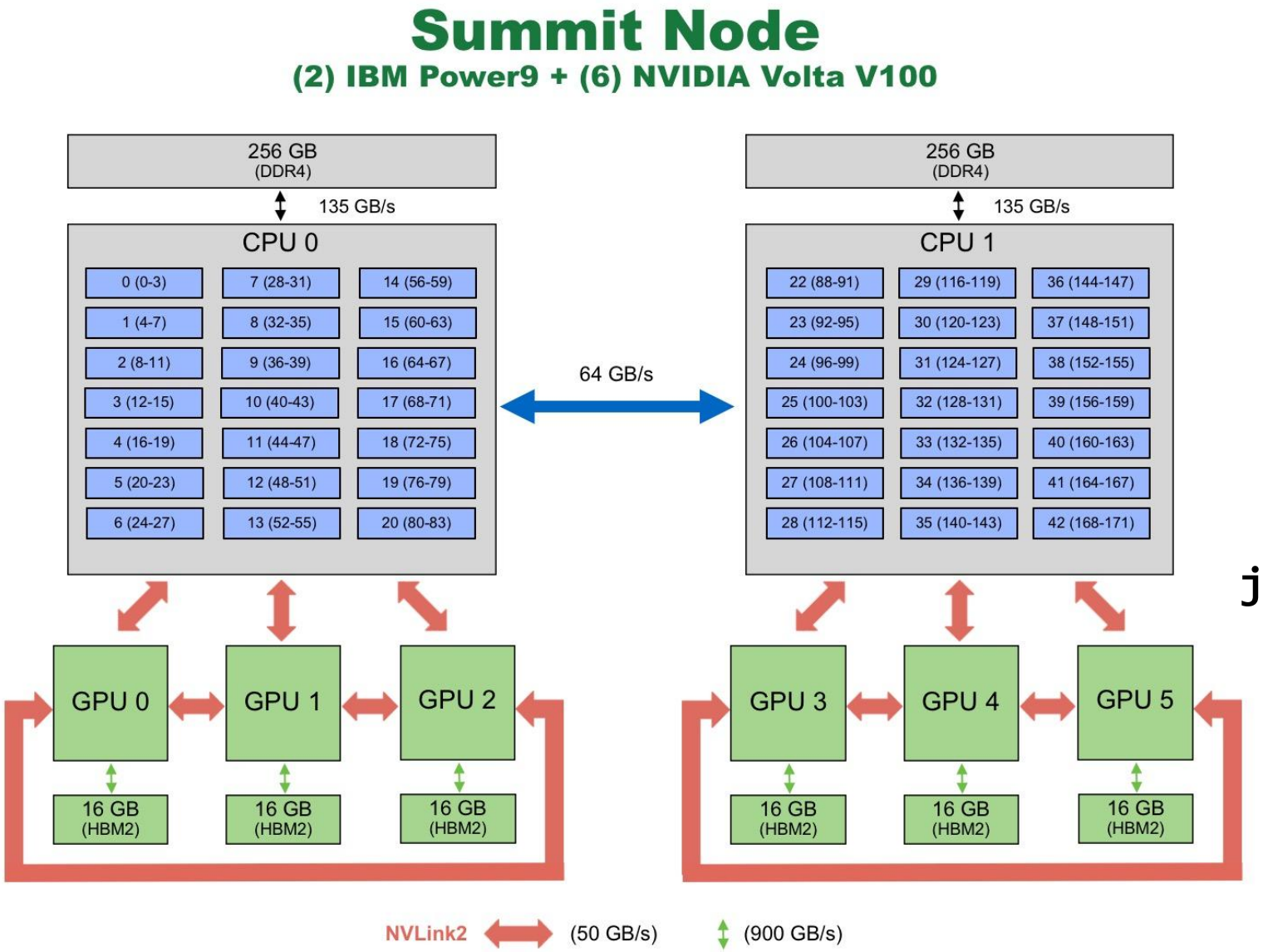
Fluid Evolution on CPU

```
 1: Call: Initialize ( )
 2: Call: GhostExchange ( )
 3: Set: Time = StartTime
 4: while Time < FinishTime do
 5:     Call: ComputeTimeStep ( ) → TimeStep
 6:     Set: FluidOld = FluidCurrent
 7:
 8:     Call: ComputeDifferences ( )
 9:     Call: ComputeReconstruction ( )
10:     Call: ComputeFluxes ( )
11:     Call: ComputeUpdate ( TimeStep ) → FluidUpdate
12:     Set: FluidCurrent = FluidOld + FluidUpdate
13:
14:     Call: GhostExchange ( )
15:
16:     Call: ComputeDifferences ( )
17:     Call: ComputeReconstruction ( )
18:     Call: ComputeFluxes ( )
19:     Call: ComputeUpdate ( TimeStep )
20:     Set: FluidCurrent = 0.5 * (FluidOld + FluidCurrent + FluidUpdate)
21:
22:     Call: GhostExchange ( )
23: end while
```

Fluid Evolution on GPU

```
 1: HOST: Call: Initialize ( )
 2: Call: GhostExchange ( )
 3: HOST: Set: Time = StartTime
 4: while Time < FinishTime do
 5:    HOST: Call: ComputeTimeStep ( ) → TimeStep
 6:    TRANSFER: Call: FluidCurrent % UpdateDevice ( )
 7:    DEVICE: Set: FluidOld = FluidCurrent
 8:
 9:    DEVICE: Call: ComputeDifferences ( )
10:    DEVICE: Call: ComputeReconstruction ( )
11:    DEVICE: Call: ComputeFluxes ( )
12:    DEVICE: Call: ComputeUpdate ( TimeStep ) → FluidUpdate
13:    DEVICE: Set: FluidCurrent = FluidOld + FluidUpdate
14:
15:    TRANSFER: Call: FluidCurrent % UpdateHost ( )
16:    HOST: Call: GhostExchange ( )
17:    TRANSFER: Call: FluidCurrent % UpdateDevice ( )
18:
19:    DEVICE: Call: ComputeDifferences ( )
20:    DEVICE: Call: ComputeReconstruction ( )
21:    DEVICE: Call: ComputeFluxes ( )
22:    DEVICE: Call: ComputeUpdate ( TimeStep )
23:    DEVICE: Set: FluidCurrent = 0.5 * (FluidOld + FluidCurrent + FluidUpdate)
24:
25:    TRANSFER: Call: FluidCurrent % UpdateHost ( )
26:    Call: GhostExchange ( )
27: end while
```

Performance Results

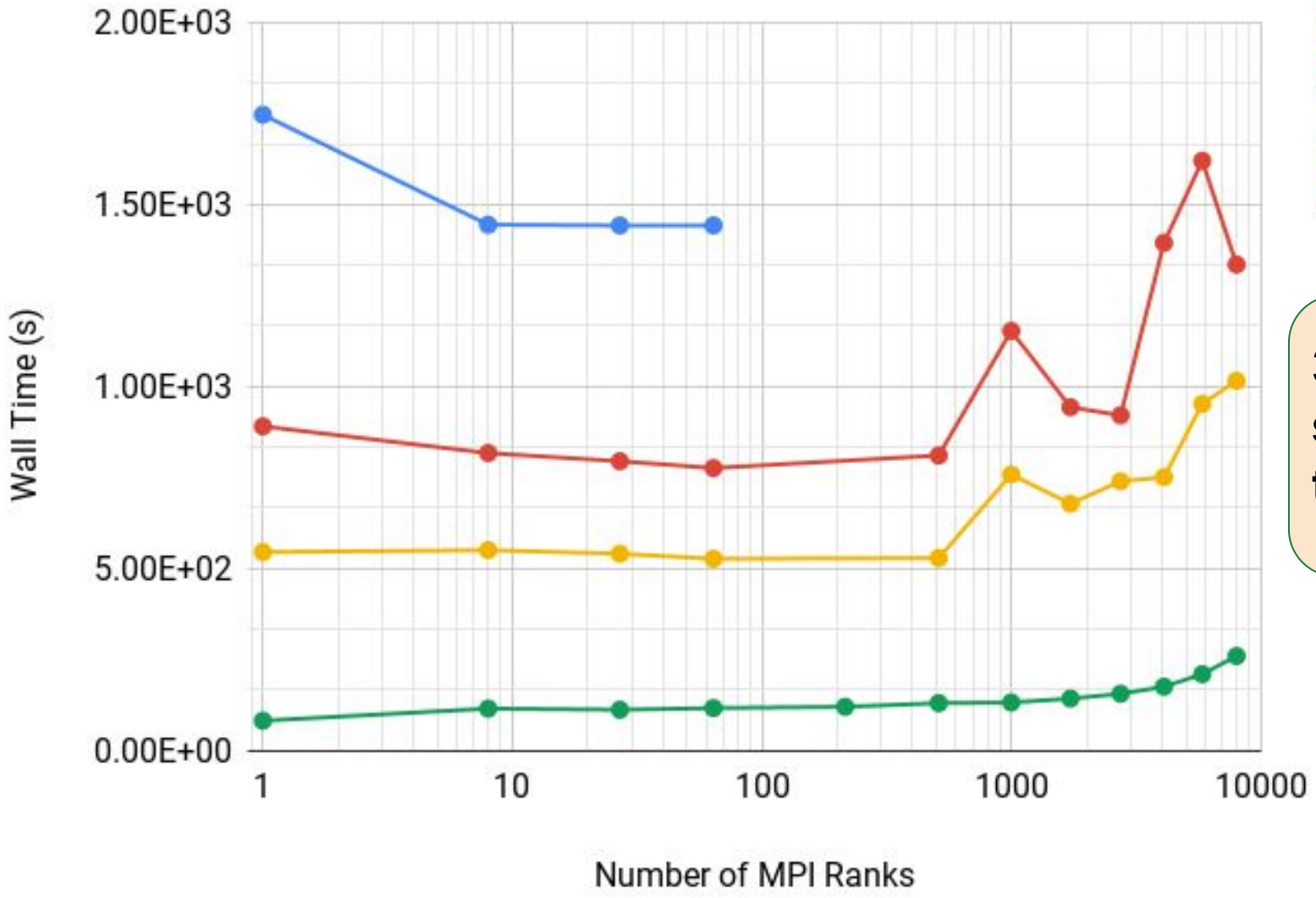


"Proportional resource tests":
7 CPU cores vs. 1 GPU

6 RS per host,
1 MPI (+OpenMP) per RS

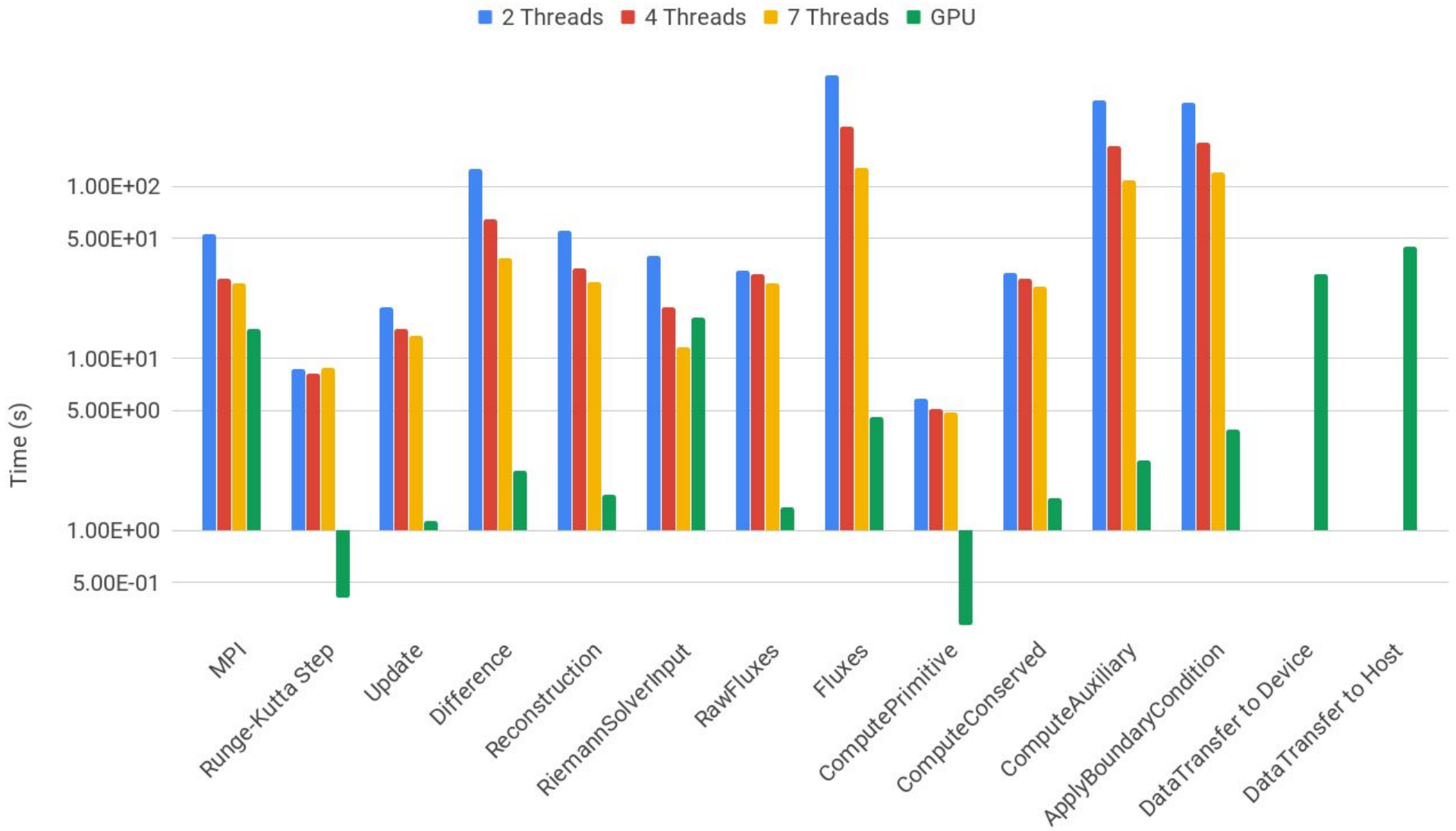
```
jsrun -r6 -c7 -g1 -a1 -bpacked:7
```

Performance Results: Weak-Scaling 3D RiemannProblem

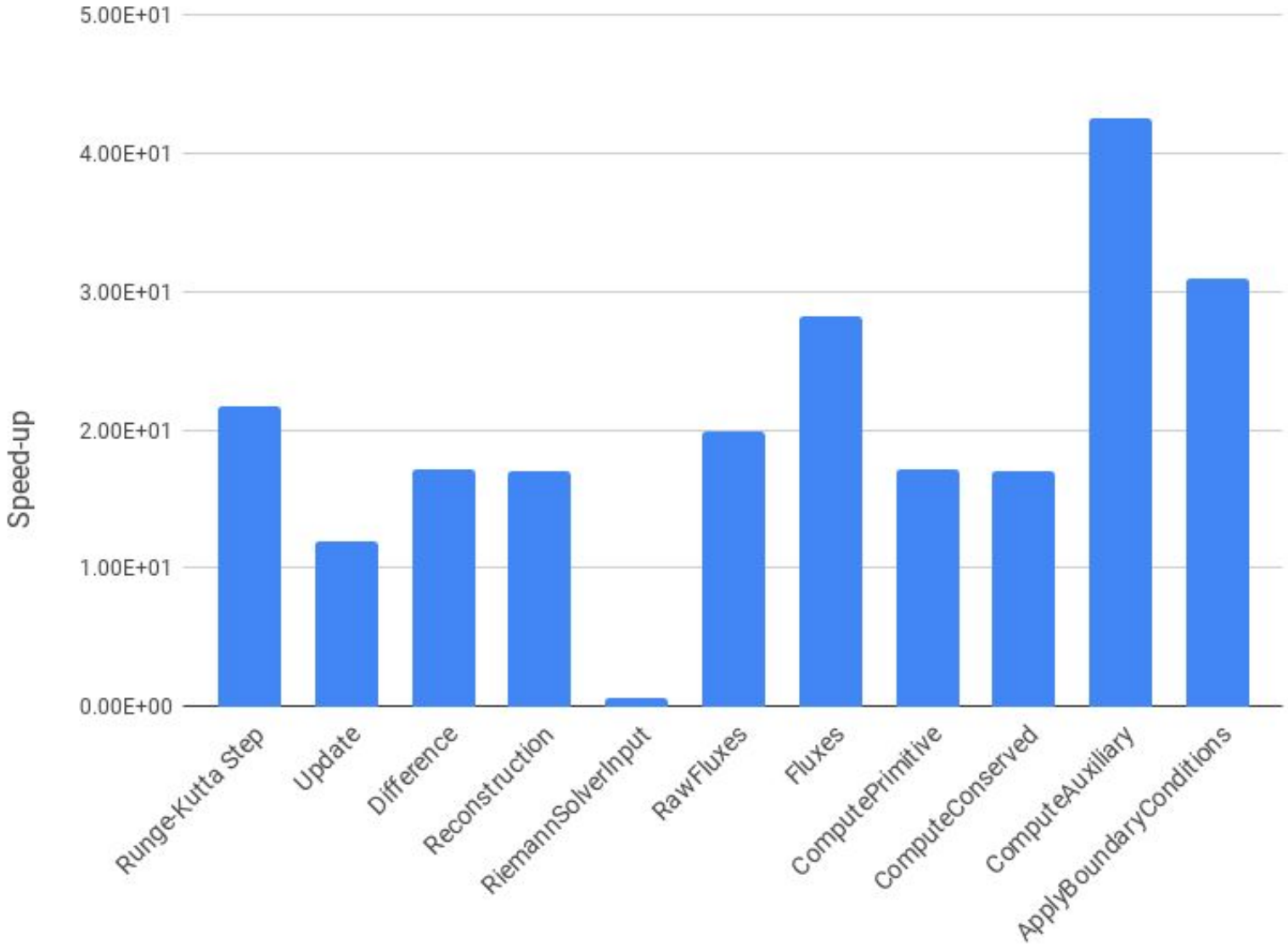


3.92X - 6.71X speedup from 7 CPU threads to GPU

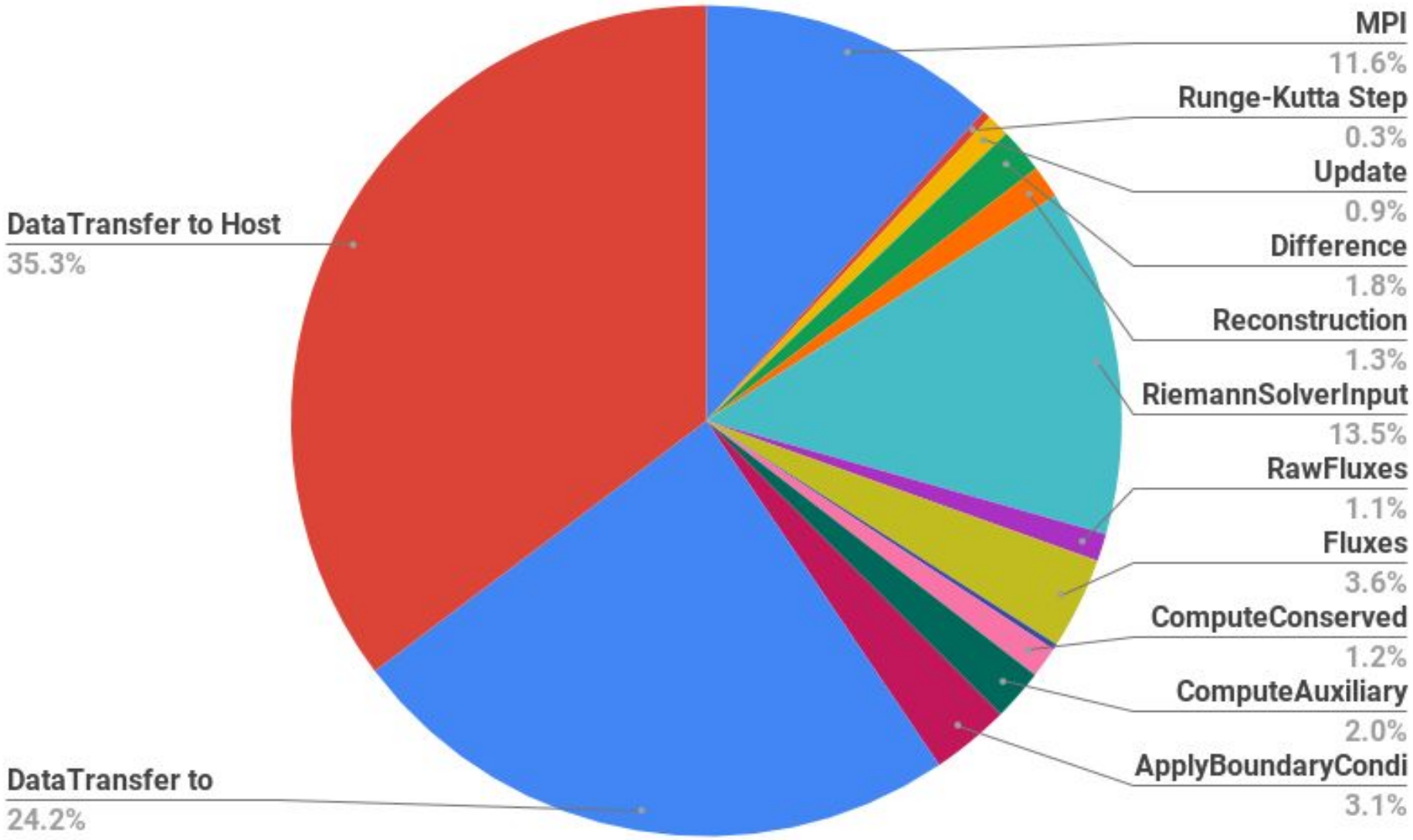
Performance Results: Kernel Timings



Performance Results: Kernel Speedups



Performance Results: Timing Distribution

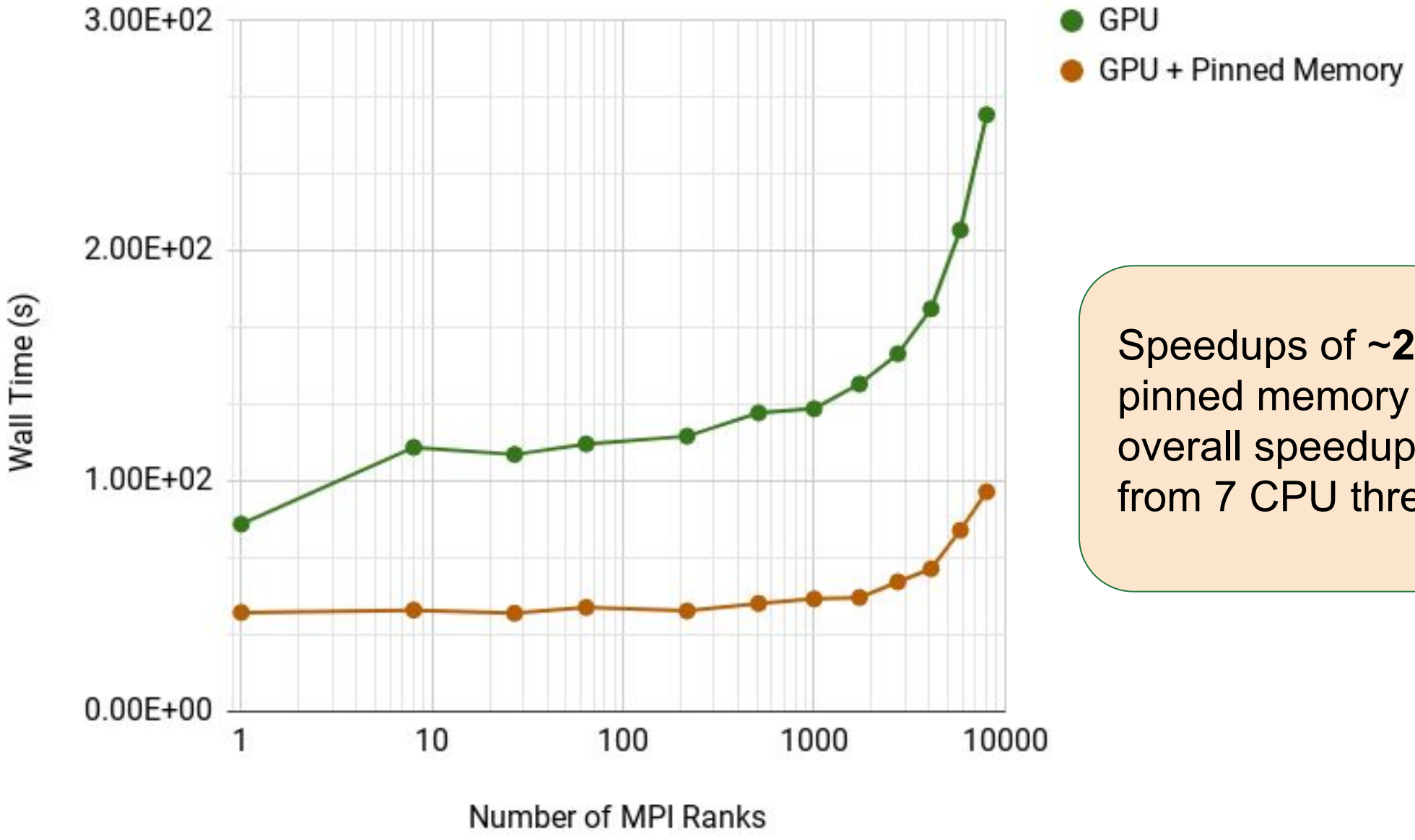


Beyond OpenMP: Using Pinned Memory

To optimize data transfers:

- pinned memory: page-locked host memory allocated using `cudaMallocHost()` or `cudaHostAlloc()`
- created another Fortran wrapper in GenASiS, used in `StorageForm` initialization method as an option
  `StorageForm % Initialize ( ..., PinnedOption = .true.)`
- No mechanism to do this with OpenMP 4.5 (but perhaps in 5.x)

Performance Results: Using Pinned Memory



Speedups of **~2.6X** when pinned memory is used → overall speedups of **over 12X** from 7 CPU threads

Implications (Then and Now)

- c. 2010: 1024^3 cells with 64000 processors (Jaguar), ~3s per timestep
  Now: 64 GPUs (11 nodes) on Summit, ~1.2s per timestep

- Enable us to do higher-fidelity simulations, ensemble studies for trends in observables
  - plan to perform ~200 2D grey transport supernova simulations, tens of 3D grey transport, and a handful of 3D spectral transport simulations

- First step towards full Boltzmann radiation transport (6-D problem + time) with exascale computing

Remaining Issues and Future Work

- A single code-base with OpenMP for multi-threading and offload
  - Fall back to multi-threading with `target if`-clause is problematic
  - `team distribute` directive introduce deleterious effects for multi-threading
- Kernel-launch parallelism and CUDA streams
  - no mechanism within OpenMP to affect and select stream
- Better compilers support for OpenMP 4.5 - 5.x
- Using CUDA-aware MPI for GPU Direct
  - benefit (vs. manual staging on host) depends on message sizes

Conclusion

- Using OpenMP allows for a simple and effective porting of Fortran code to target GPU
  - 2 - 3 months “walltime” efforts for this project
  - ~1 day / week “person time” efforts
- OpenMP 4.5 (and later) is a path to port code to GPU
  - more compilers are supporting OpenMP offload (XL, GCC, CCE, Intel, PGI, LLVM-based)
- Code available on http://github.com/GenASiS
  - paper describing this work in preparation