Refactoring the MPS/University of Chicago Radiative MHD (MURaM) Model for GPU/CPU Performance Portability Using OpenACC Directives

Eric Wright[1], Damien Przybylski[2], Matthias Rempel[3], Cena Miller[3], Supreeth Suresh[3], Shiquan Su[3], Richard Loft[3], Sunita Chandrasekaran[1]

[1] University of Delaware, Newark, Delaware, USA
[2] Max Planck Institute for Solar System Research, Gottingen, Germany
[3] National Center for Atmospheric Research, Boulder, Colorado, USA

OpenACC Summit 2021

Acknowledgement

- **Cheyenne** (doi:10.5065/D6RX99HX) supercomputer and **Casper** data analysis and visualization cluster provided by NCAR's Computational and Information Systems Laboratory, sponsored by the National Science Foundation (NSF).
- **Cobra** supercomputer system of the Max Planck Computing and Data Facility (MPCDF) in Garching, Germany.
- **NSF under Cooperative Agreement No.1852977**
- European Research Council (ERC) under the European Union's **Horizon 2020** research and innovation programme (grant agreement No. 695075)
- **NASA's SDO/AIA (NNG04EA00C)** contract
- **OpenACC** organization for their technical support

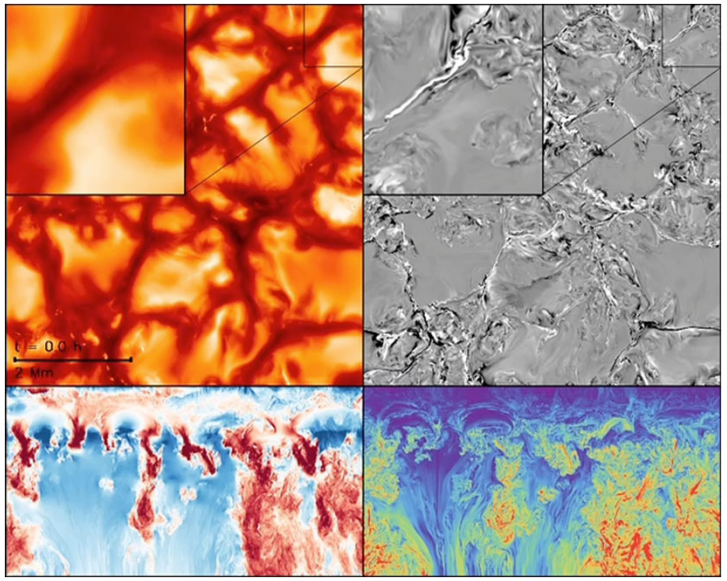
Outline

- Description of MURaM
- Challenges porting MURaM to large scale systems
- Results

DESCRIPTION OF MURAM

MURaM (Max Planck University of Chicago Radiative MHD)

- Primary solar model for simulations of the upper convection zone, photosphere and corona
- Jointly developed by HAO, the Max Planck Institute for Solar System Research (MPS) and the Lockheed Martin Solar and Astrophysics Laboratory (LMSAL)

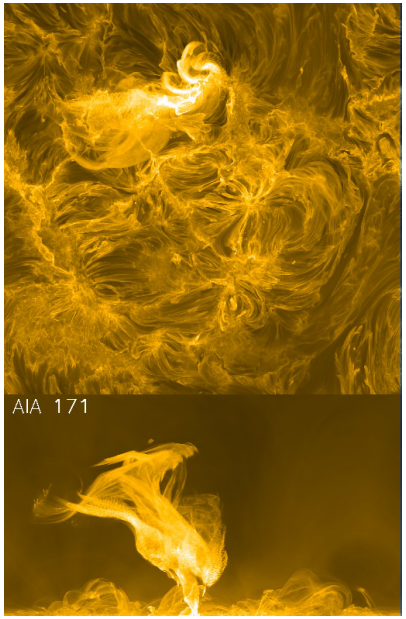


MURaM simulation of solar granulation

MURaM Future Goals

https://nso.edu/telescopes/dki-solar-telescope/

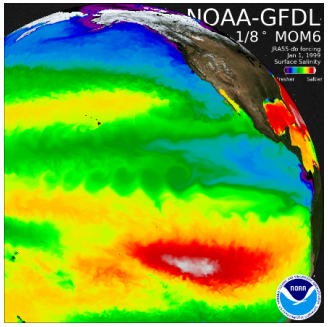
- Short-term: Solar models capable of running at higher resolutions
  - Higher resolution simulations requires 10-100x more computing time
  - Difficult to accomplish as current CPU runs are hitting a scaling limit
- Long-term: achieve real-time simulation to predict solar events
  - Current models are 10-100x slower than real-time



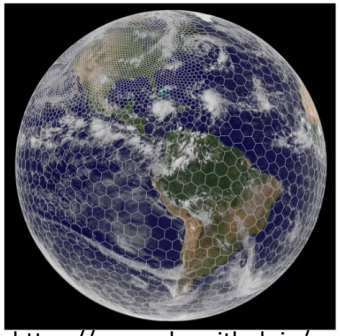
MURaM simulation: Synthetic coronal emission (SDO/AIA) during M2 Flare and resulting CME

GPU Programming with OpenACC

- Our team has experience working on several other large OpenACC projects
- Some MURaM users will prefer to run the CPU version
- Parallelize the code incrementally, without large rewrites at a time

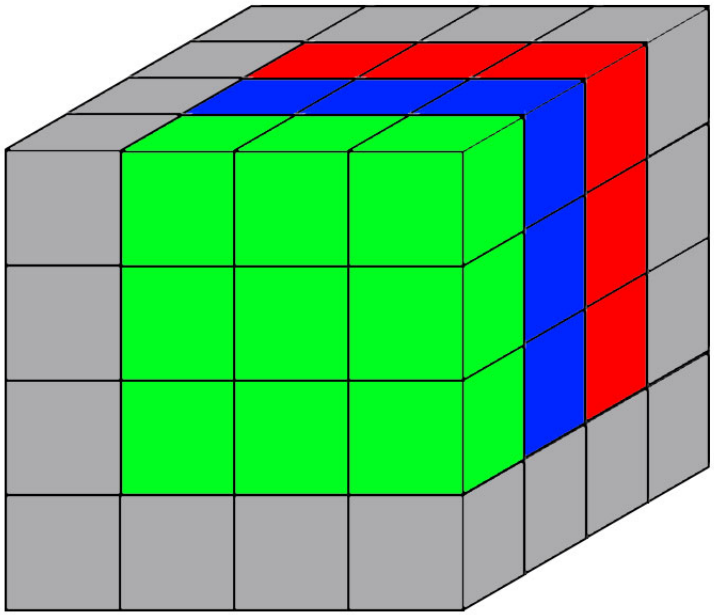


https://www.gfdl.noaa.gov/ocean-model/


https://mpas-dev.github.io/

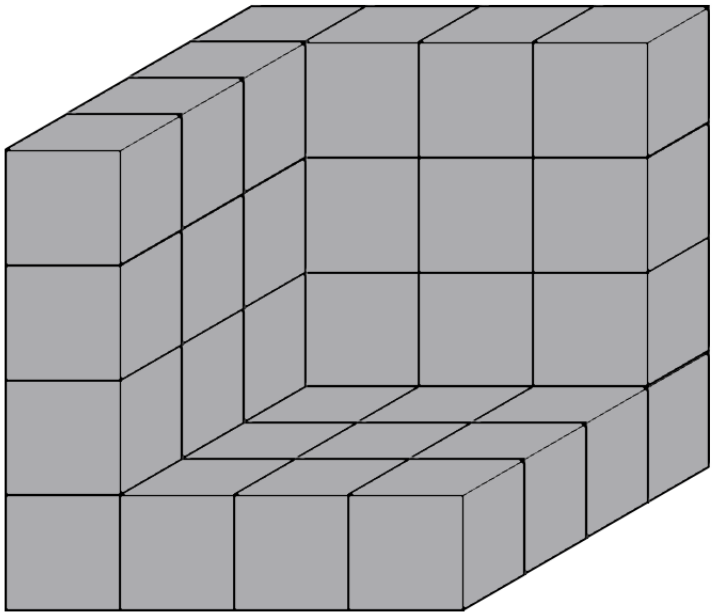
CHALLENGES PORTING MURAM TO LARGE SCALE SYSTEMS

Optimizing RTS (Radiation Transport Solver)



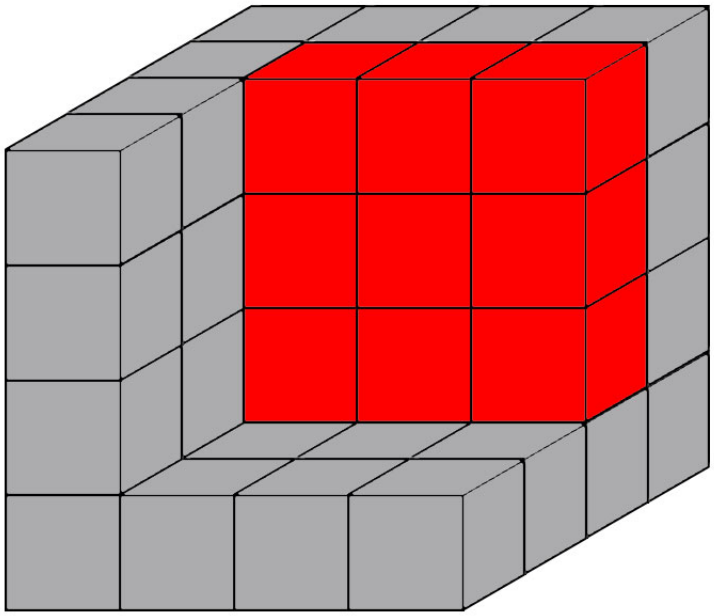
- RTS accounts for the majority of the runtime
- Within RTS, the *Integrate()* function is the most time consuming
- *Integrate()* must be run at least once for each of the 24 rays
- *Integrate()* introduces a data dependency that must be considered carefully when parallelizing

Optimizing RTS (Radiation Transport Solver)



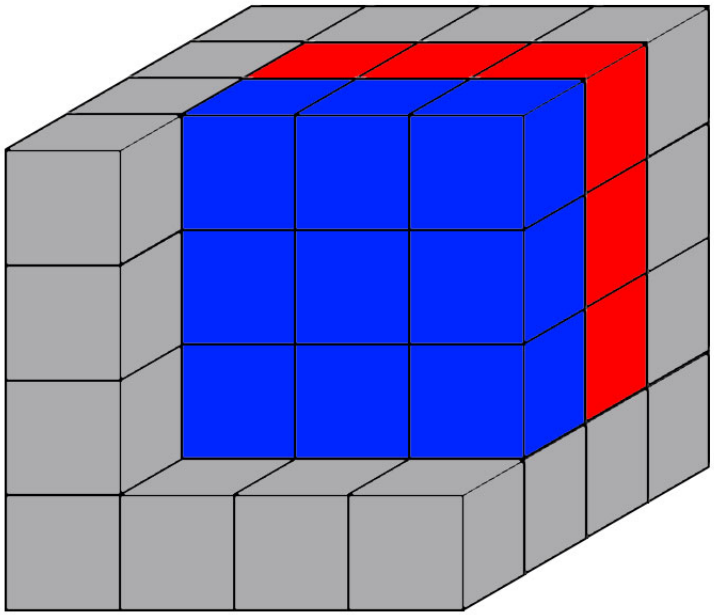
- RTS accounts for the majority of the runtime
- Within RTS, the *Integrate()* function is the most time consuming
- *Integrate()* must be run at least once for each of the 24 rays
- *Integrate()* introduces a data dependency that must be considered carefully when parallelizing

Optimizing RTS (Radiation Transport Solver)



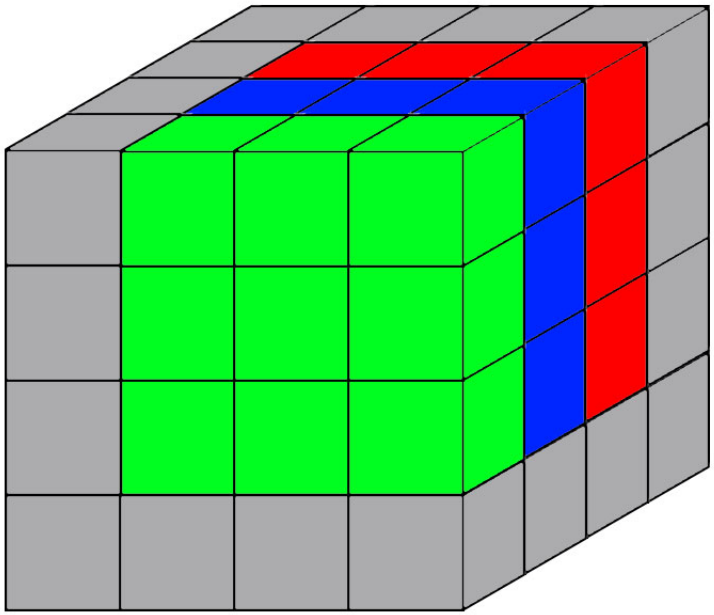
- RTS accounts for the majority of the runtime
- Within RTS, the *Integrate()* function is the most time consuming
- *Integrate()* must be run at least once for each of the 24 rays
- *Integrate()* introduces a data dependency that must be considered carefully when parallelizing

Optimizing RTS (Radiation Transport Solver)



- RTS accounts for the majority of the runtime
- Within RTS, the *Integrate()* function is the most time consuming
- *Integrate()* must be run at least once for each of the 24 rays
- *Integrate()* introduces a data dependency that must be considered carefully when parallelizing

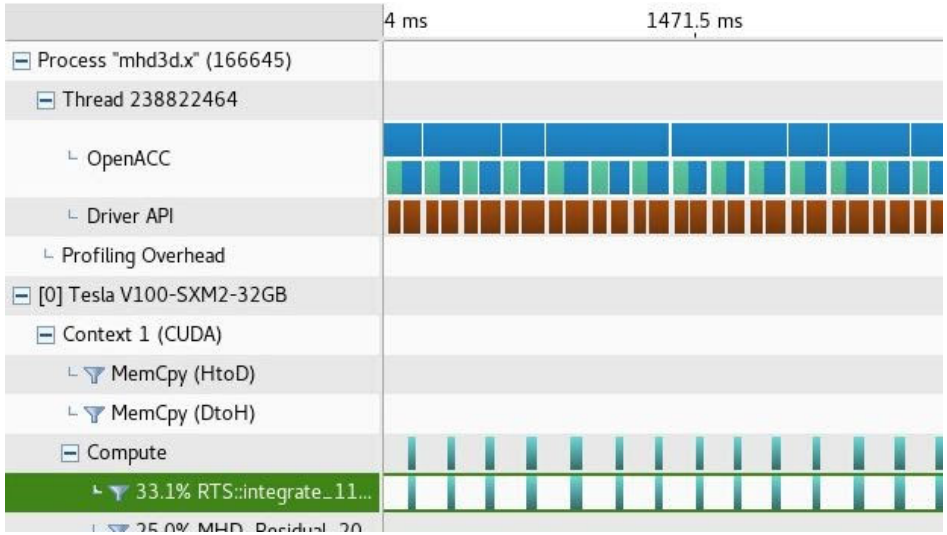
Optimizing RTS (Radiation Transport Solver)



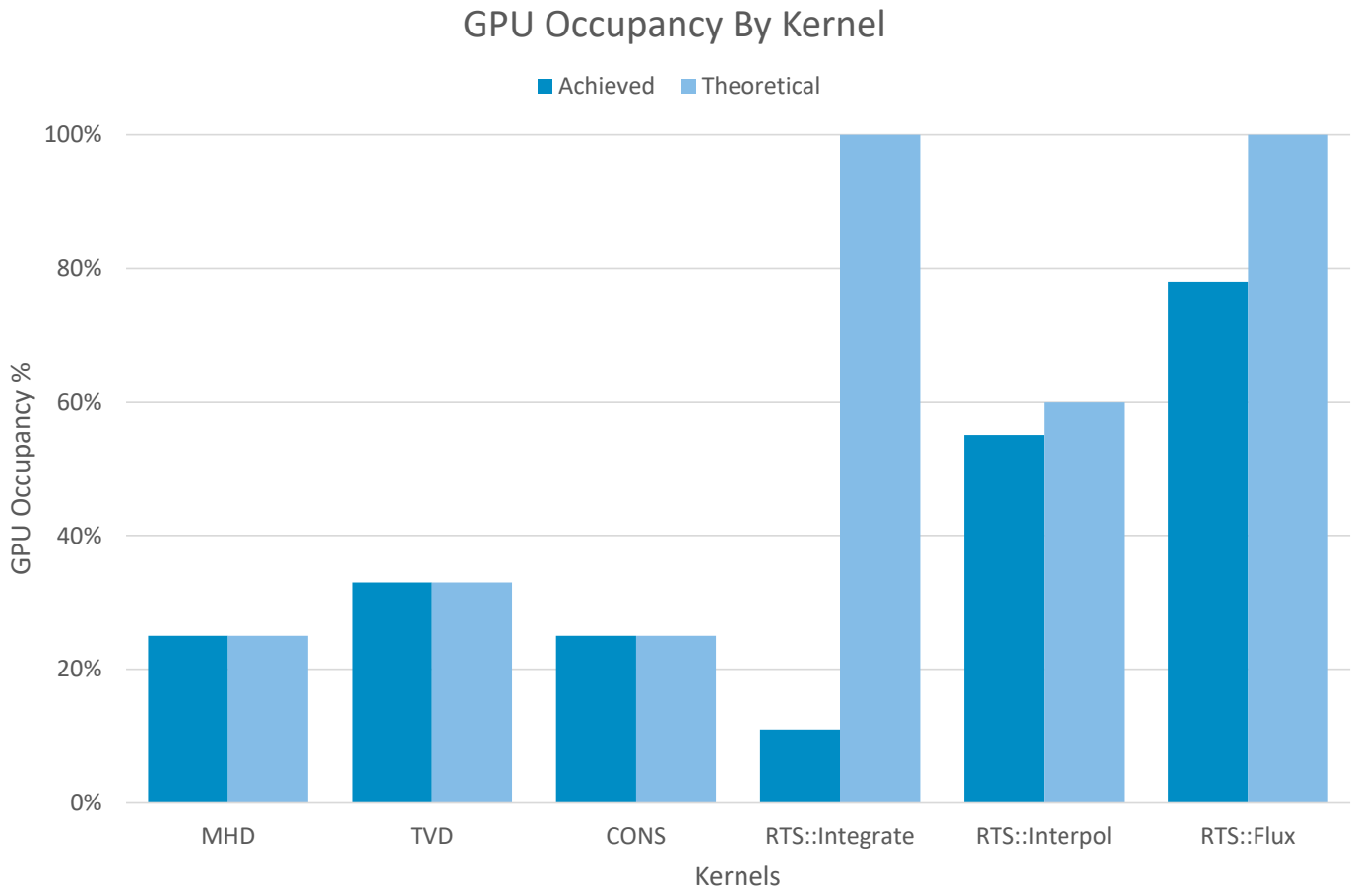
- RTS accounts for the majority of the runtime
- Within RTS, the *Integrate()* function is the most time consuming
- *Integrate()* must be run at least once for each of the 24 rays
- *Integrate()* introduces a data dependency that must be considered carefully when parallelizing

Optimizing RTS: Problems with *Integrate()*

- The outermost loop over the grid-cells must be sequential
  - This causes many (hundreds) of GPU kernels to be launched for each call to Integrate
  - These GPU kernels only parallelize 2 dimensions of our grid-space, resulting in small, inefficient kernels (with low GPU occupancy)



Code Performance GPU Occupancy



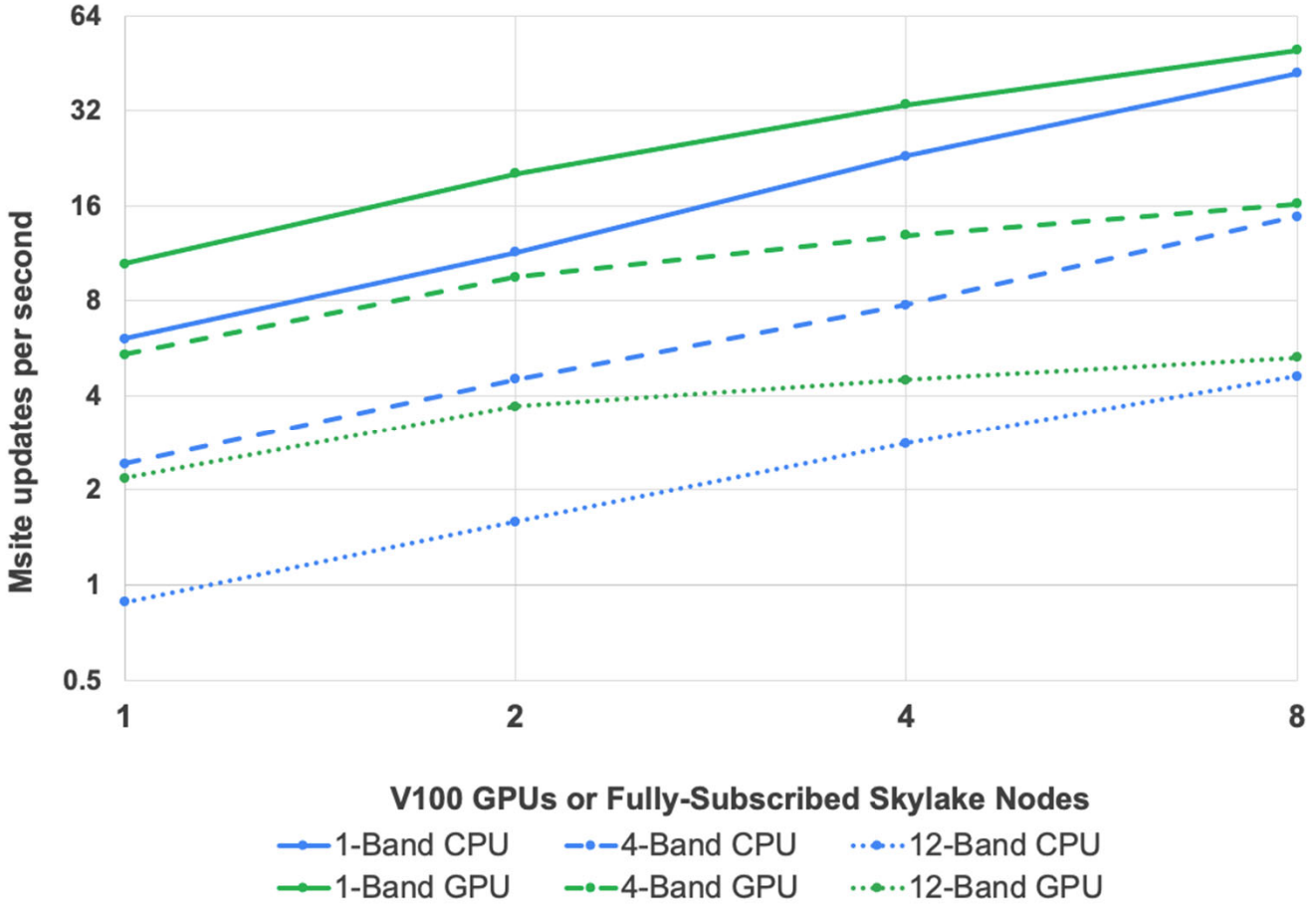
RESULTS

Experimental Setup: Cobra

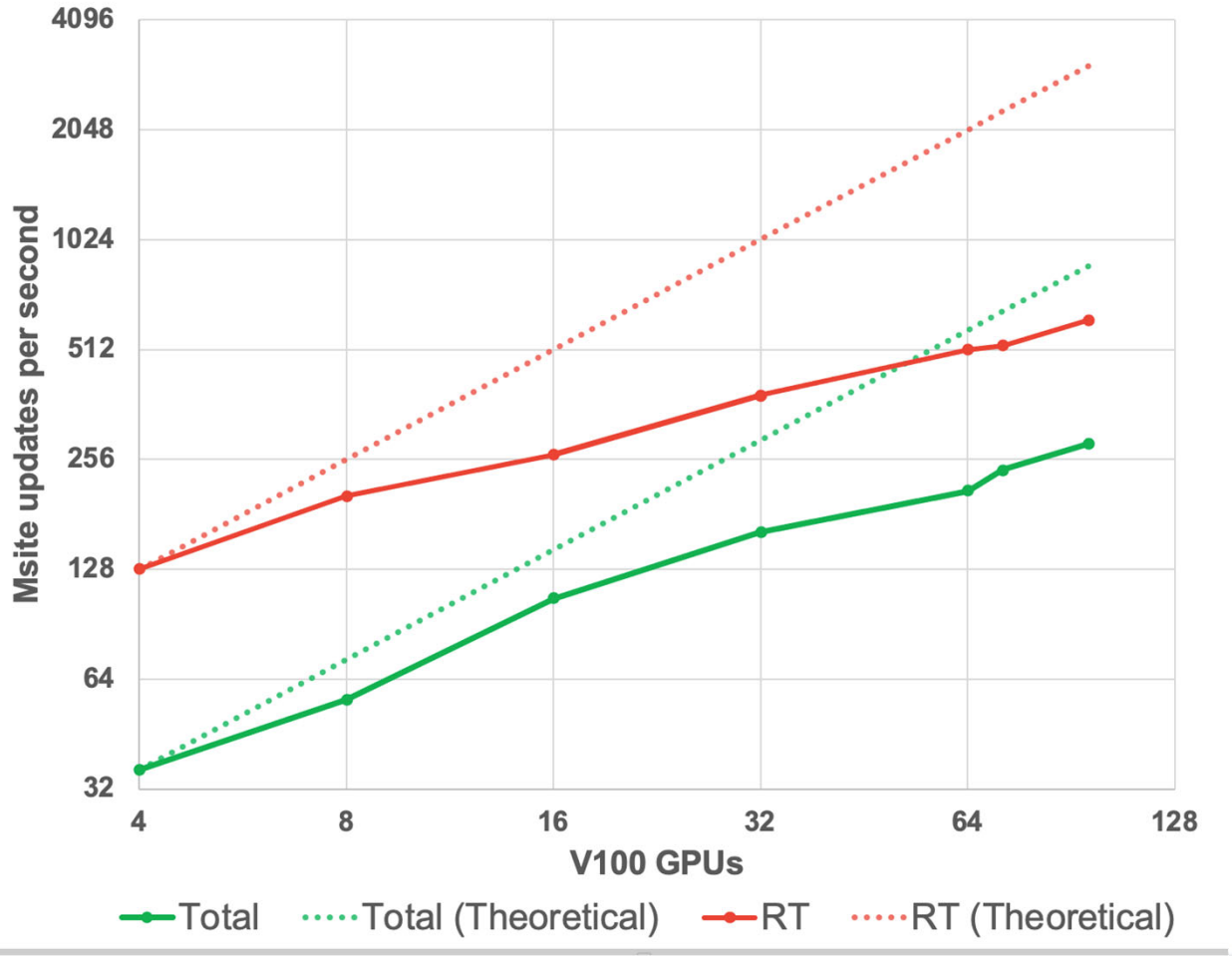
Max Planck Computing and Data Facility (MPCDF) in Garching, Germany

- 3,424 compute nodes
  - two Intel Xeon Gold 6148 Skylake (SKL) processors (20 cores at 2.4GHz)
  - 100 Gb/s Omni Path interconnect
- 64 GPU nodes
  - two NVIDIA Tesla V100-PCIe with 32GB HBM2
- CPU compilers/libs: Intel 19.1.3, Intel MPI 2019.9, MKL 2020.2 and FFTW-MPI 3.3.8
- GPU compilers/lib: NVHPC 20.9, CUDA 11, OpenMPI 4.0.5 with UCX 1.8.0 and FFTW-MPI 3.3.8 with multithreading

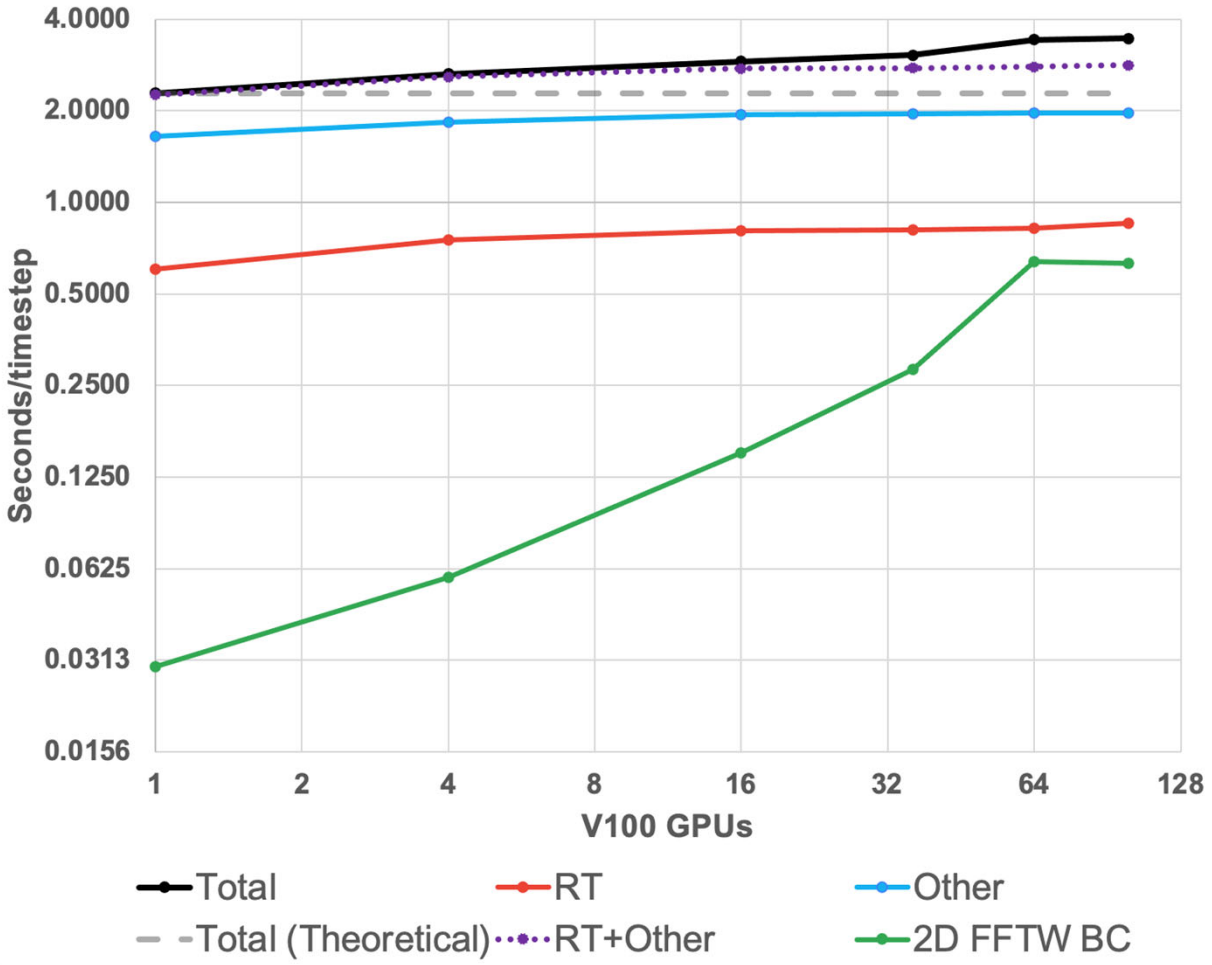
Strong Scaling: 288^3 dataset on 8 GPUs



Strong Scaling: 288x576x576 dataset on up to 96 GPUs



Weak Scaling: 288^3 datapoints per GPU



Summary

- 1.73x speedup using a single NVIDIA V100 GPU over a fully subscribed 40-core Intel Skylake CPU node
- Maintained single source code across multicore CPUs and GPUs
- Multi-band radiation transport will advance understanding of the solar chromosphere

Future Work

- Implementation of ray-merge version of the code
- Explore GPU-enabled FFT libraries (heFFTe)
- Implement ensemble capabilities to get closer to real-time prediction

Published with ACM PASC 2021: https://dl.acm.org/doi/10.1145/3468267.3470576