Performance Portability is (sort of) a Lie

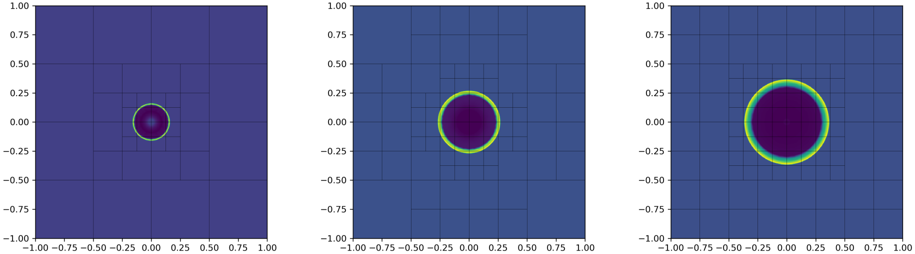
Jonah Miller

Los Alamos National Laboratory

Salishan Meeting for High-Speed Computing

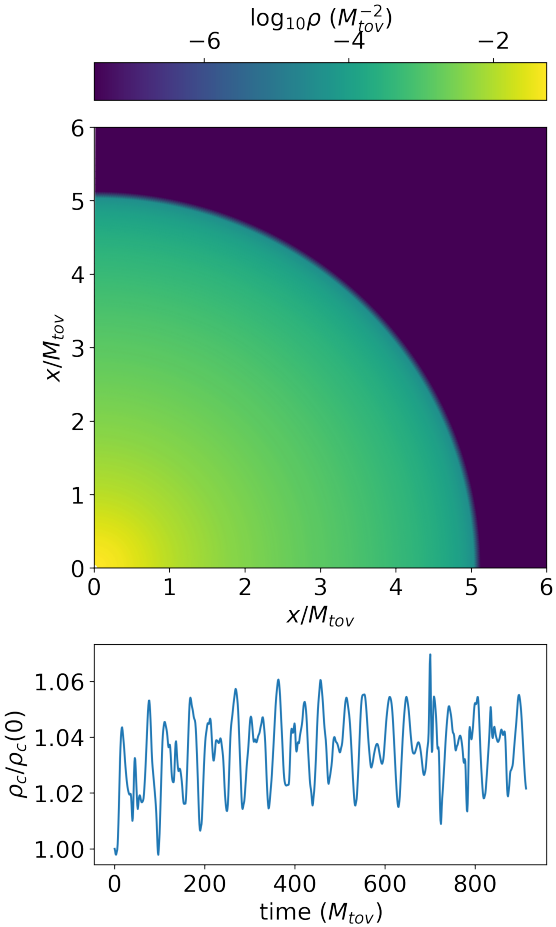
Parthenon

- Growing international collaboration
- *Meshblocks* appear to be "uniform grids" on which applications can be built:
  - Memory locality easier to enforce
  - Complexity can be "lifted" out of inner loops
  - Convenient serializable data structure for tasking
  - Convenient prototyping platform
- Join us: `https://github.com/lanl/parthenon`

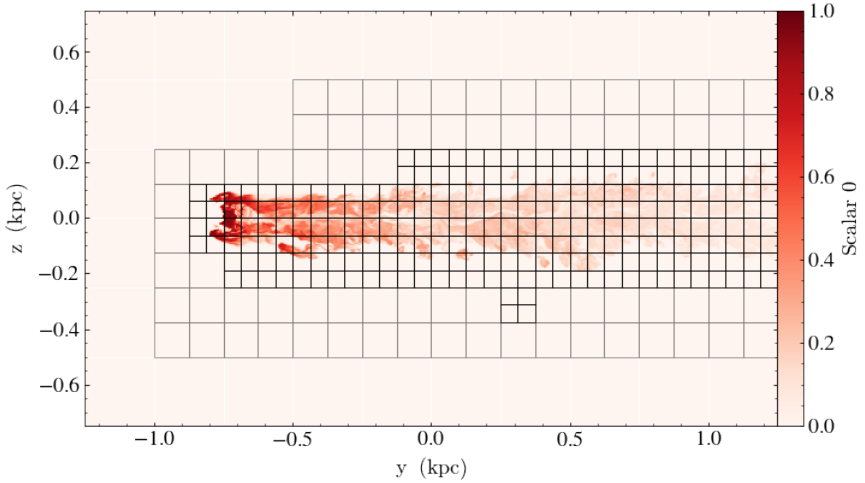


Applications/Downstream Codes

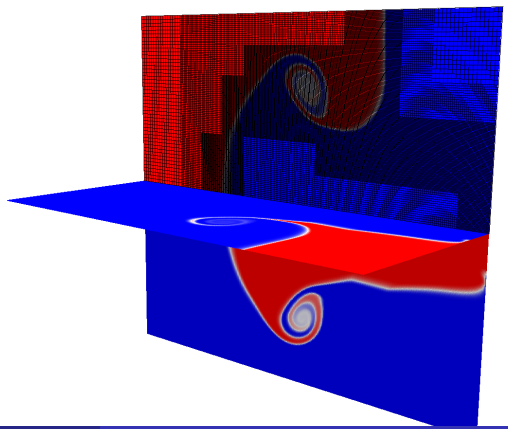
- Phoebus, neutron star



- Athena-PK, turbulence

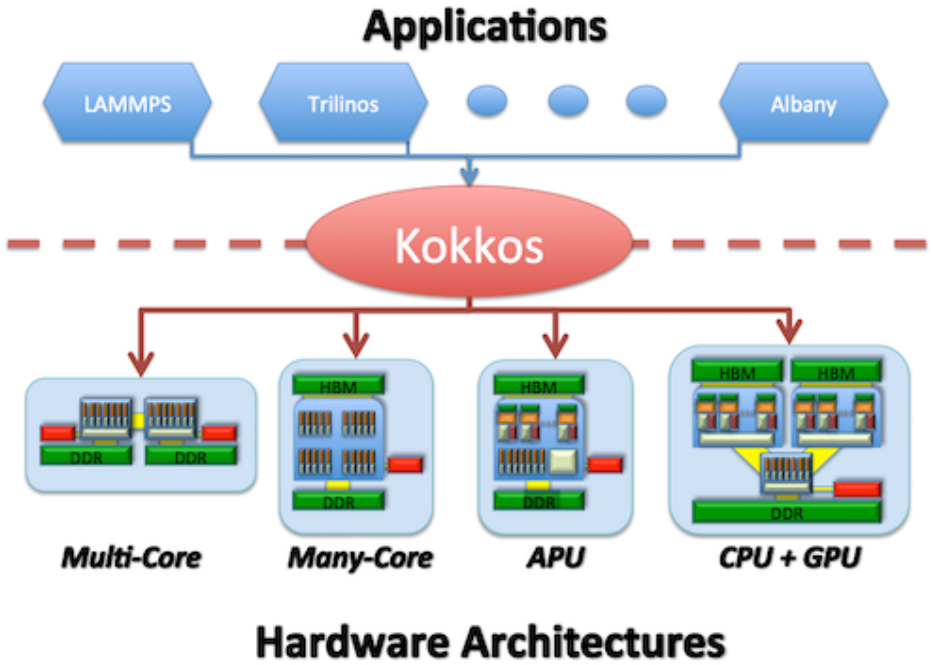


- Riot, triple point



Kokkos and Performance Portability

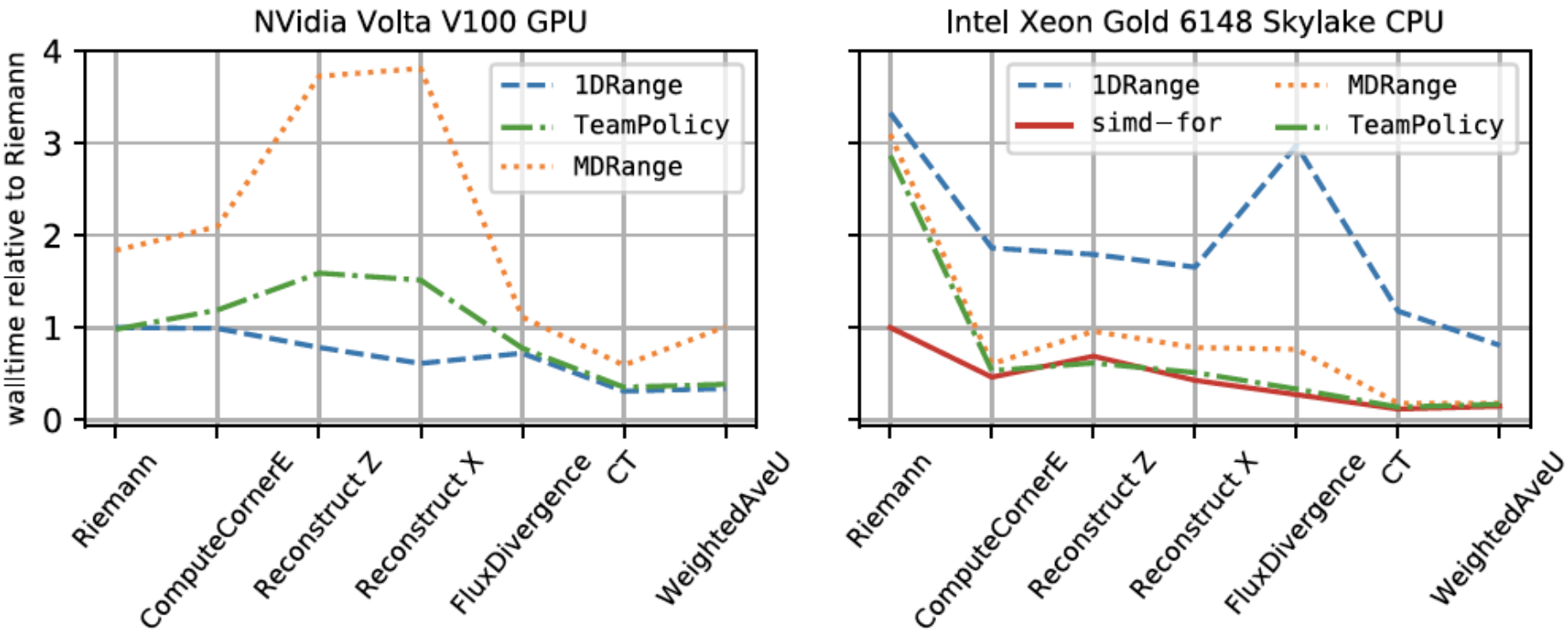
- Parthenon uses *Kokkos* for performance portability
- Select target architecture at compile time, templates specialize to resolve to code for target architecture



Edwards, Trott, and Sunderland, 2014

Looping, Memory Locality, Vectorization

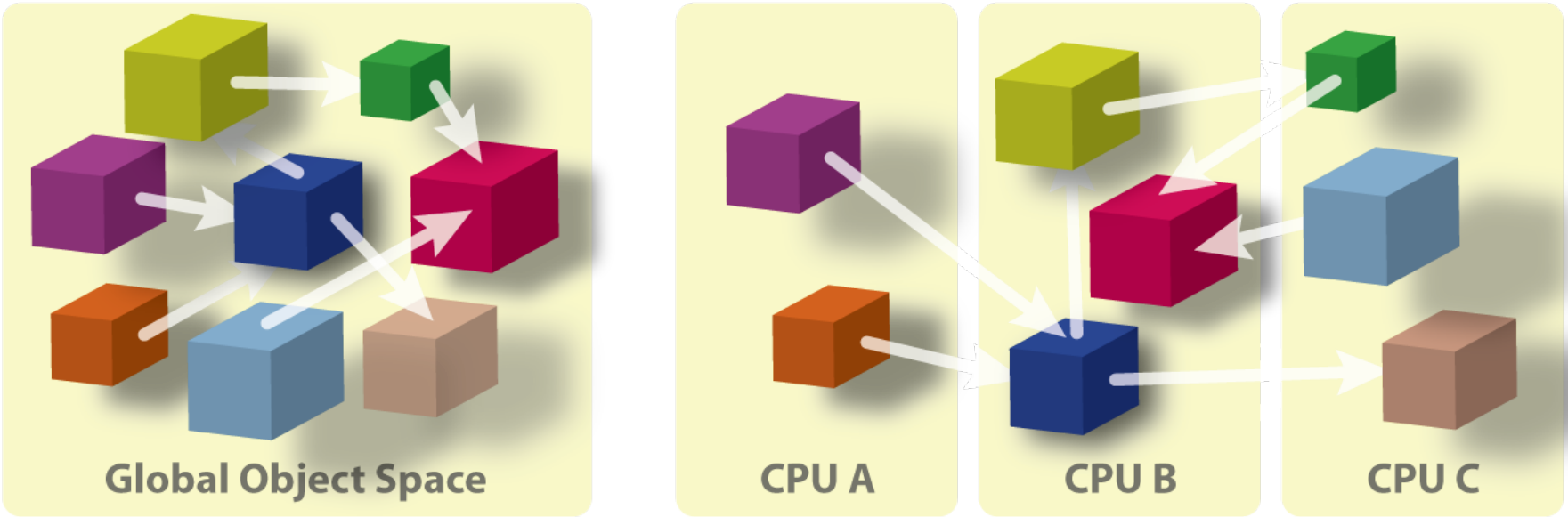
- Kokkos exposes different loop patterns, such as manually flattened, hierarchical parallelism, etc
- These patterns have significant effect on performance
- On CPU, hand-optimized simd loops are fastest. On GPU, manually unrolled 1D loops.



Grete, Glines, O'Shea, 2021

Asynchronicity Vs. Latency

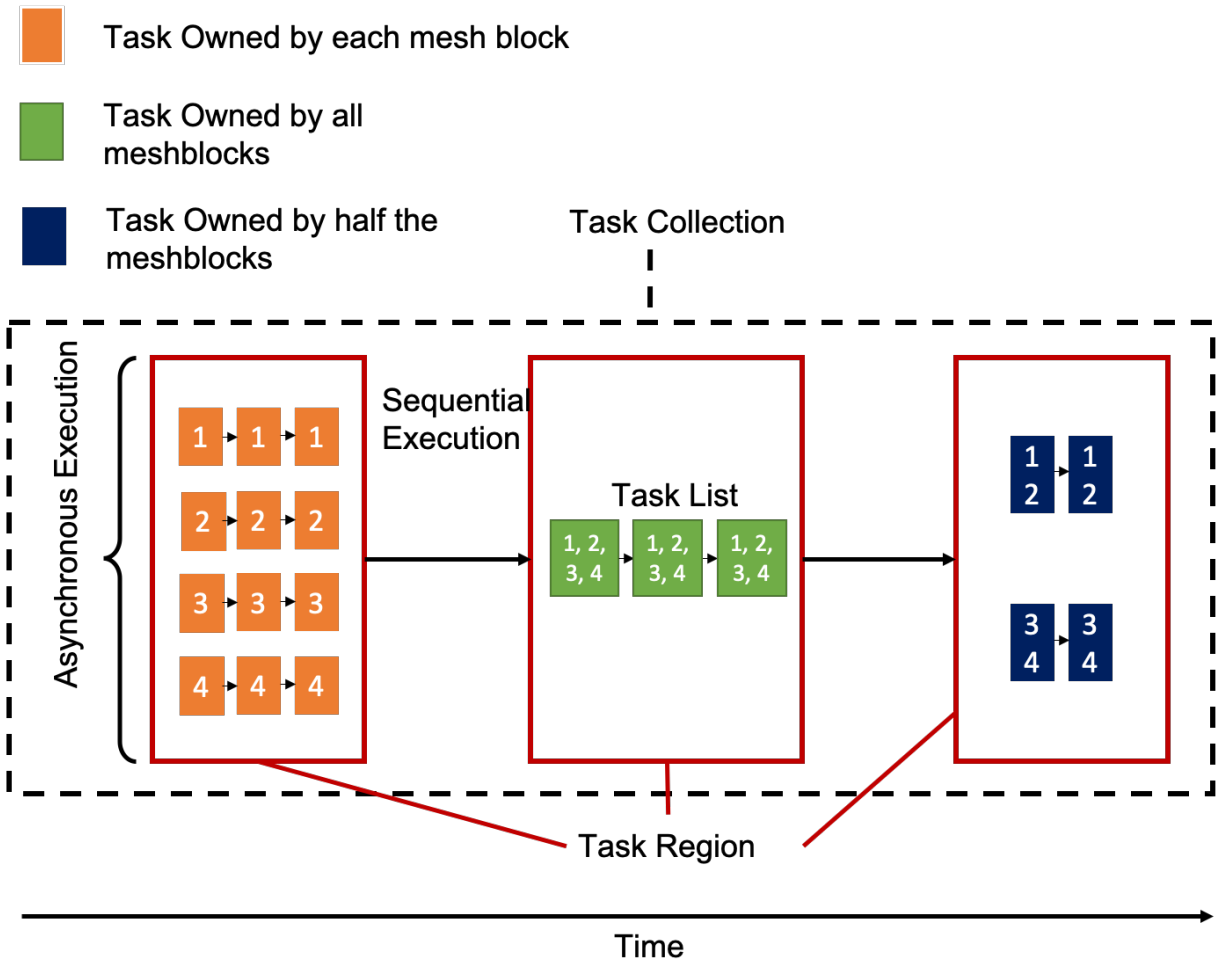
- To handle load imbalance, task-based framework expresses work in *tasks*, which are localized to operate on pieces of data
- Requires *overdecomposition* so that tasks/data can be moved accross cores/workers
- In Parthenon, tasks operate on *meshblocks*, which are discrete pieces of the mesh
- **Problem:** Launching a task costs $\sim$6 μs on GPU



Charm++ Team

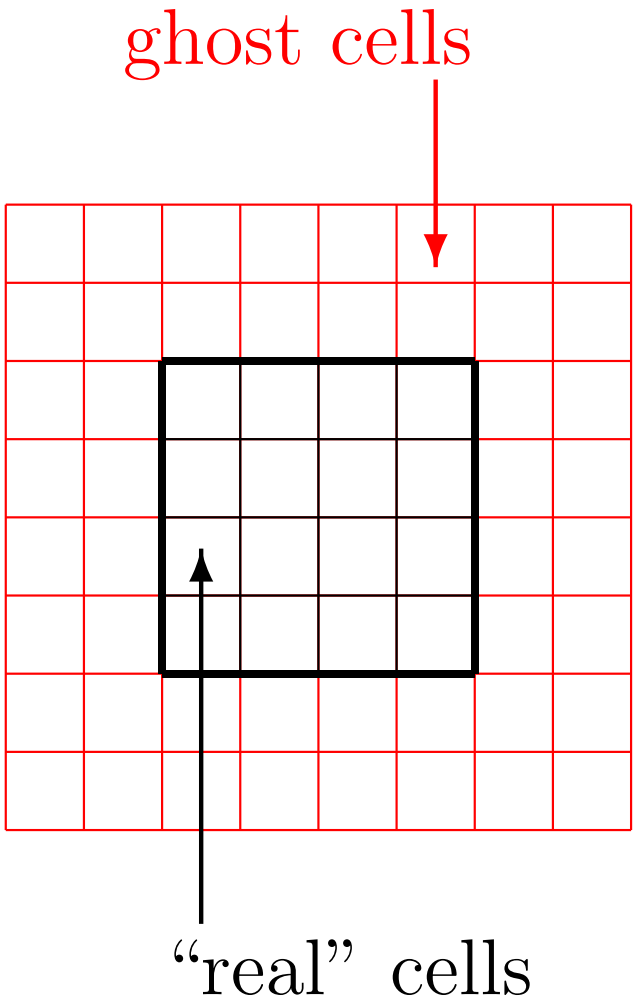
Algorithmic Change—Meshblockpacks

- **Solution:** Coalesce GPU kernels accross meshblocks
- This comes at a price: coalescence exposes less asynchronicity
- Number of coalesced kernels is run-time tunable and can be programmed *per-task*

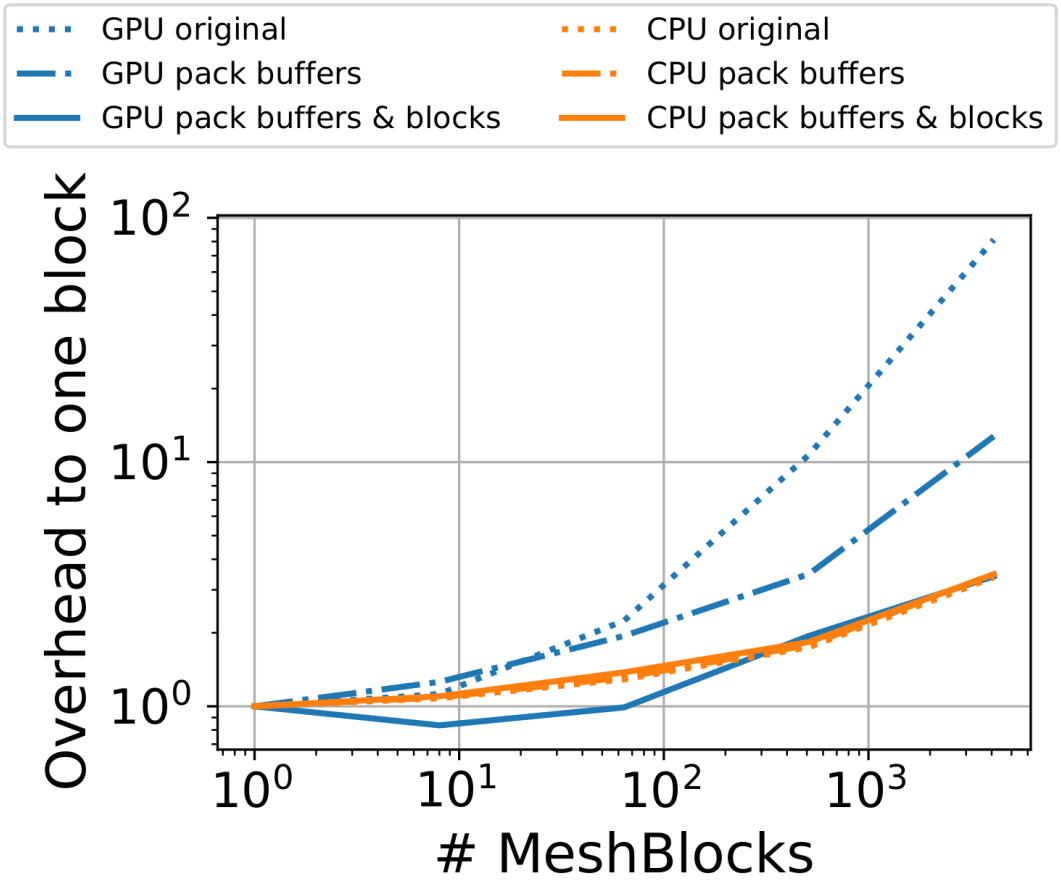


Grete, **JMM**, et al., ArXiv:2202.12309

Measured Overhead



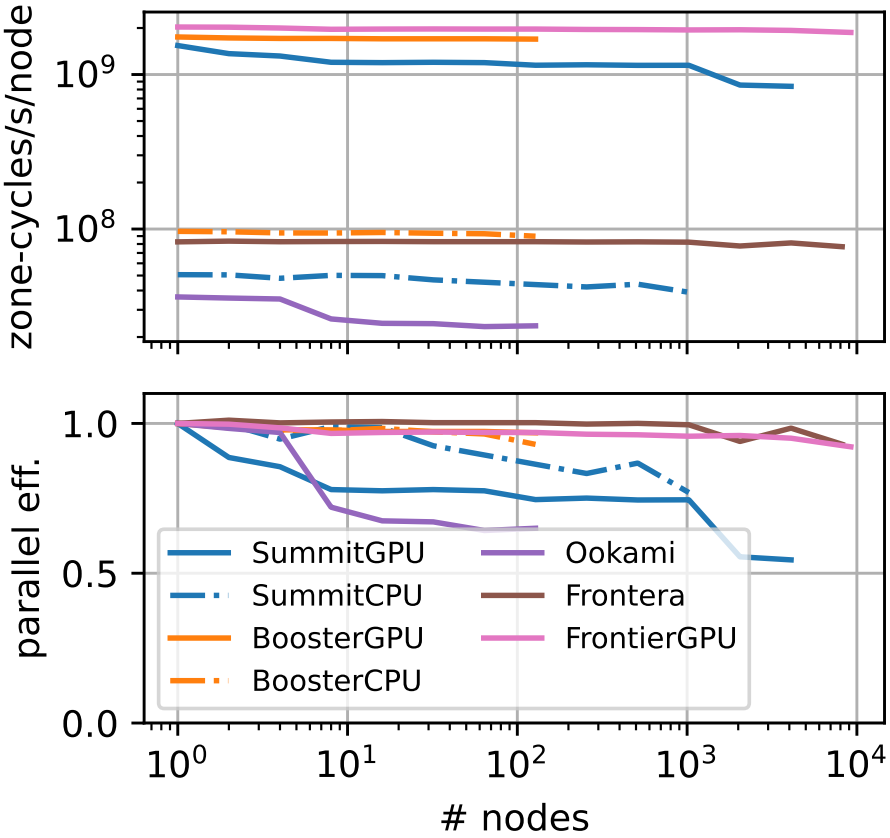
- Measured decomposition of 256^3 grid



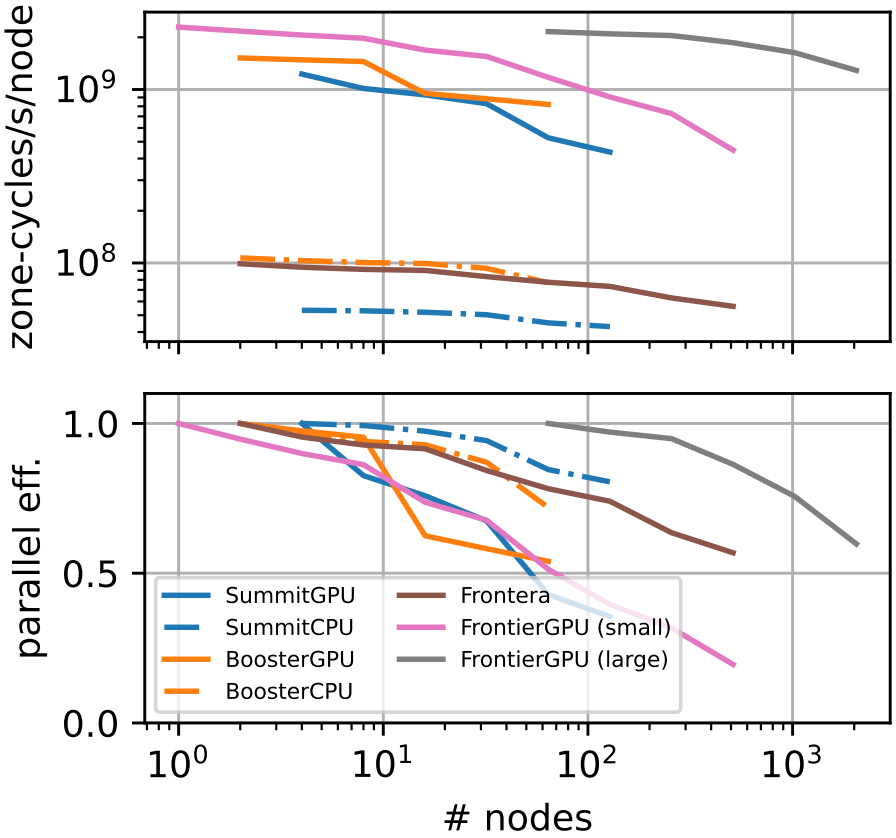
Grete, **JMM**, et al., ArXiv:2202.12309

Scaling

- Weak Scaling



- Strong Scaling



Grete, **JMM**, et al., ArXiv:2202.12309

Performance Portability is (Sort of) a Lie

- A bet against a software stack in flux.
- Enables code that compiles on lots of architectures with little effort.
- But just because your code compiles, doesn't mean it's fast!
- True performance requires thinking about the *algorithm* and tuning it to the hardware you're running on.
- We found a great deal of success exposing tuning parameters within the infrastructure.
- Great wins to be had, both on-node and at scale.

Where you can find this work

- This work is mostly from a series of two papers:
  - arXiv:1905.04341v2
  - arXiv:2202.12309
  - arXiv:2206.08957
  - More on the way!
- Many authors contributed to these works. The work of a collaboration is presented here.
- Questions? Want to get involved?
  - Email: `jonahm@lanl.gov`
  - Github: `https://github.com/lanl/parthenon`
  - Chat room: `https://app.element.io/#/room/#parthenon-general:matrix.org`

Applications/Downstream Codes

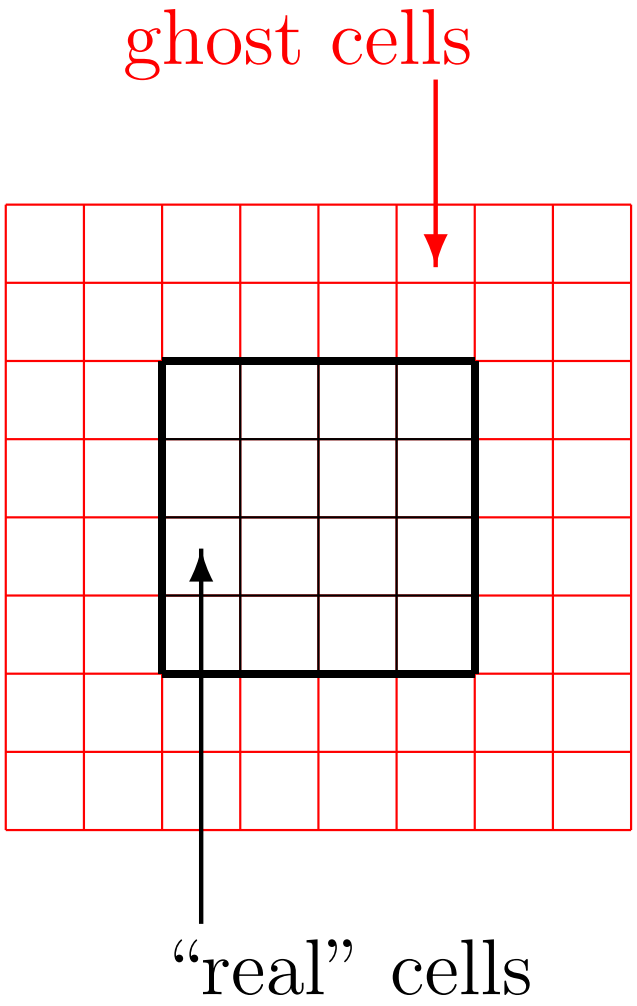
Codes built on top of or inspired by Parthenon (that we know of):

- `parthenon-hydro` (MSU): Toy Fluid Solver in about 1000 lines, `https://github.com/pgrete/parthenon-hydro`
- `Athena-PK` (MSU): General-Purpose MHD solver based on Athena++, `https://gitlab.com/theias/hpc/jmstone/athena-parthenon/athenapk`
- `KHARMA` (UIUC): Accretion disks around black holes, `https://github.com/AFD-Illinois/kharma`
- `AthenaK` (Princeton): General relativistic radiation MHD, targeting AGN disks, X-ray binaries. Not built on top of Parthenon, but heavily inspired by it.
- `Phoebus` (LANL): General relativistic radiation MHD, targeting supernova, nuetron star mergers, `https://github.com/lanl/phoebus`
- `Riot` (LANL): Multi-material fluid dynamics for engineering applications. Treats "real" materials, like dry air, steel, and wood.

Advantages of Loop Abstractioon

- We abstract out our loops, beyond the Kokkos abstraction
- We can select Kokkos-based patterns, as well as hand-written patterns
- Can manually call out to simd, openmp, Cuda, SYCL, etc
- Very flexible!
- More importantly can pick the loop pattern most performant for a given piece of hardware
- Algorithmic details, such as loop pattern, must be varied accross architectures!

How Good Can We Possibly Do?



- Ghost cells needed for communication between meshblocks.
- Ghost cells filled at every stage by MPI communication, direct copy, prolongation, and restriction, depending on mesh topology.
- Number of layers depends on algorithm used. Classic Gudonov needs 1 layer, but high-order finite differences or constrained transport may need as many as 4 layers.
- Overhead, ratio of total to real cells:

$$O_v(N) = \frac{(N + 2G)^3}{N^3}$$

- If N large, $O_v(N) \to 1$
- As $N \to 0$, $O_v(N) \to \infty$

In General, More MPI Ranks/Device is Better

- Performance in 10^8 zone-cycles/second on 16 Summit nodes for fixed mesh sizes and various options to distribute workload
- Base grid: $2048 \times 1536 \times 1024$ GPU, $1792 \times 384 \times 256$ CPU
- Multilevel mesh has refinened box with side length 0.4 (base grid has side length 1)
- Multiple MPI ranks per GPU enabled with NVIDIA MPS
- Behaviour maybe explained by heterogeneity—CPUs can do infrastructure work while GPUs do physics work.

	Uniform mesh							Multilevel mesh			
# blocks per dev.	1	2		16				259 (GPU) & 37 (CPU)			
# packs per rank	1	1	B	1	2	4	B	1	2	4	B
1 rank per GPU	10.8	11.7	10.7	11.7	11.3	11.0	9.1	2.2	2.2	2.2	1.0
2 ranks per GPU	–	12.9	–	12.6	12.6	12.2	11.6	2.9	3.0	3.0	1.7
4 ranks per GPU	–	–	–	13.0	13.1	12.9	12.9	3.9	4.0	4.0	2.7
1 rank per CPU core	0.45	0.47	0.44	0.25	0.29	0.29	0.29	0.42	0.43	0.42	0.40

Grete, **JMM**, et al., ArXiv:2202.12309

Performance Portability

- In the end we found good performance accross a wide range of devices and architectures
- Typical uniform grid workload in Parthenon-Hydro, in 10^8 zone-cycles/device-second:

Device (Arch./Instr.)	Performance
NVIDIA A100 GPU (CUDA Cap. 8.0)	4.2
NVIDIA V100 GPU (CUDA Cap. 7.0)	2.7
AMD MI100 GPU (ROCm)	2.15
AMD EPYC 7H12 (2x64C x86 AVX2)	1.45
Intel Xeon 6148 (2x20C x86 AVX512)	0.67
Intel Xeon E5-2680v4 (2x14C x86 AVX2)	0.43
Fujitsu A64FX (1x48C ARMv8.2-A)	0.36

Grete, **JMM**, et al., ArXiv:2202.12309

Of Course, Each Machine is Unique

Machine	Node conf.	Environment	Compiler optimization flags
Summit GPU	2x 22-core Power9 CPU, 6x V100 16GB, NVLink, 2x EDR InfiniBand	GCC 9.1.0, CUDA 11.0.3, SpectrumMPI 10.4.0.3	`-O3 -mcpu=power9 -mtune=power9 -expt-extended-lambda -Wext-lambda-captures-this -arch=sm_70`
Summit CPU			`-O3 -mcpu=power9 -mtune=power9 -fopenmp-simd -fprefetch-loop-arrays`
Booster GPU	2x 24-core Epyc 7402 CPU, 6x A100 40GB, NVLink3, 4x HDR200 InfiniBand	GCC 11.2.0, CUDA 11.5, OpenMPI 4.1.1	`-O3 -march=znver2 -mtune=znver2 -expt-extended-lambda -Wext-lambda-captures-this -arch=sm_80`
Booster CPU			`-O3 -march=znver2 -mtune=znver2 -fopenmp-simd -fprefetch-loop-arrays`
Spock GPU	1x 64-core EPYC 7662, 4x MI100, Infinity Fabric (xGMI), Slingshot-10	HIP 4.4.21401, ROCm 4.5.0, Cray MPICH 8.1.12	`-O3 -march=znver2 -mtune=znver2 -fno-gpu-rdc --amdgpu-target=gfx908`
Spock CPU			`-O3 -march=znver2 -mtune=znver2 -fopenmp-simd -fprefetch-loop-arrays`
Ookami	1x 48-core Fujitsu A64FX, 1x HDR200 InfiniBand	Fujitsu FCC 4.5.0, OpenMPI 4.0.1	`-Nclang -O3 -ffj-fast-matmul -ffast-math -ffp-contract=fast -ffj-fp-relaxed -ffj-ilfunc -fbuiltin -fomit-frame-pointer -finline-functions -ffj-preex -ffj-zfill -ffj-swp -fopenmp-simd`

Grete, **JMM**, et al., ArXiv:2202.12309

Other Optimizations Performed

- Coalesce memory allocations, especially at mesh refinement
- Optimize loop structure, especially for buffer packing
- Data structures that ensure memory locality
- To encourage vectorization, minimize branching and indirection