First experiences at the exascale with Parthenon – a performance portable block-structured adaptive mesh refinement framework

Philipp Grete

Hamburg Observatory

in collaboration with the Parthenon community (J. Dolence, F. Glines, J. Miller, P. Mullen, B. Prather, B. Ryan, L. Roberts, J. Stone, and more) and J. Holmen (OLCF)

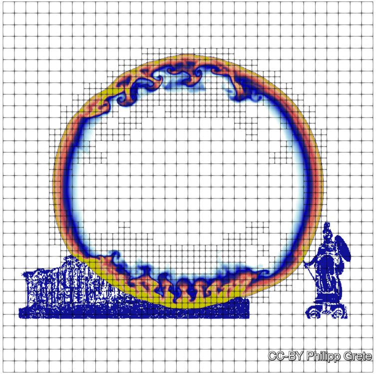
January 2024 OLCF User Conference Call

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Funded by the European Union

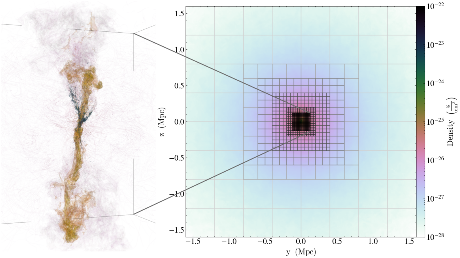
(Adaptive) Mesh Refinement (AMR)

- Decompose domain into blocks
- Blocks
  - are logically independent
  - have fixed size
  - communicate with their neighbor through ghost cells/buffer zones
- "Refine" (split block into more blocks) to
  - increase spatial resolution in region(s) of interest
  - save computational resources
- Block size is important
  - ratio of active to passive zones
  - number of neighbors
  - thickness of transition regions



Parthenon – Performance portable AMR framework

- Open collaboration (10+ active developers)
- AMR framework heavily expanded from Athena++
- Intermediate abstraction layer hiding Kokkos
- Key performance design decisions
  - device first/resident
  - block packing
  - device-to-device communication via one-sided, async. MPI
- Advanced features (e.g., abstract data containers, package system, task-based parallelism, sparse variables)
- Multiple downstream codes
  - AthenaPK (MHD), Phoebus (GRMHD), KHARMA (GRMHD), parthenon-hydro (miniapp)

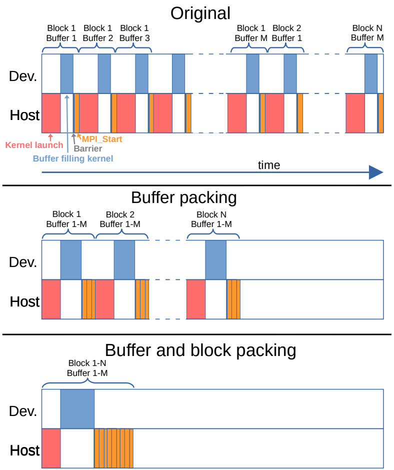


Packing #1: Kernel fusing $\leftrightarrow$ block packing

[Grete+ IJHPCA 2023 – Parthenon collaboration]

- Launch overhead
  - $\approx 5\mu$s launch, inherently serial (launching in parallel does not help)
  - possibly $> 100,000$ buffers per device
- Small blocks $\Rightarrow$ little work
  - $16^3 = 4$k cells $\leftrightarrow$ >1k cores/device
  - even Riemann solve is $< 5\mu$s

$\Rightarrow$ Combine work into fewer kernels

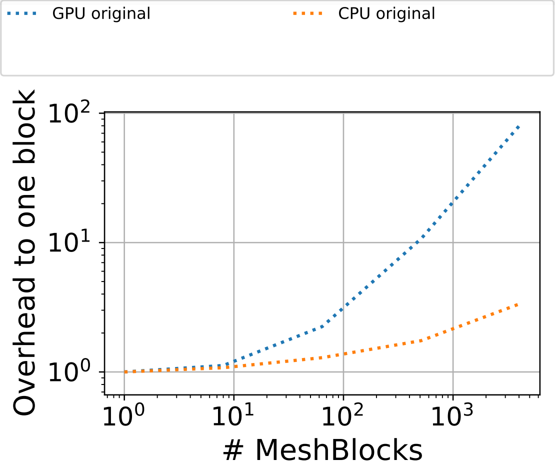


Packing #1: Kernel fusing ↔ block packing

[Grete+ IJHPCA 2023 – Parthenon collaboration]

- Launch overhead
  - $\approx 5\mu$s launch, inherently serial (launching in parallel does not help)
  - possibly $> 100,000$ buffers per device
- Small blocks $\Rightarrow$ little work
  - $16^3 = 4$k cells ↔ >1k cores/device
  - even Riemann solve is $< 5\mu$s

$\Rightarrow$ Combine work into fewer kernels



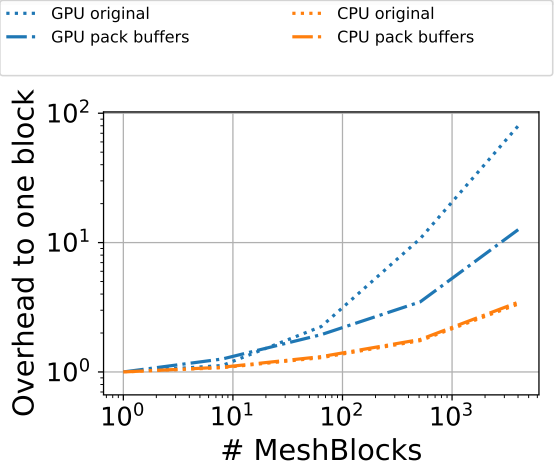
- GPU 256^3 mesh with blocks 256^3 to 16^3
- CPU 128^3 mesh with blocks 128^3 to 8^3

Packing #1: Kernel fusing $\leftrightarrow$ block packing

[Grete+ IJHPCA 2023 – Parthenon collaboration]

- Launch overhead
  - $\approx 5\mu$s launch, inherently serial (launching in parallel does not help)
  - possibly $> 100,000$ buffers per device
- Small blocks $\Rightarrow$ little work
  - $16^3 = 4$k cells $\leftrightarrow$ >1k cores/device
  - even Riemann solve is $< 5\mu$s

$\Rightarrow$ Combine work into fewer kernels



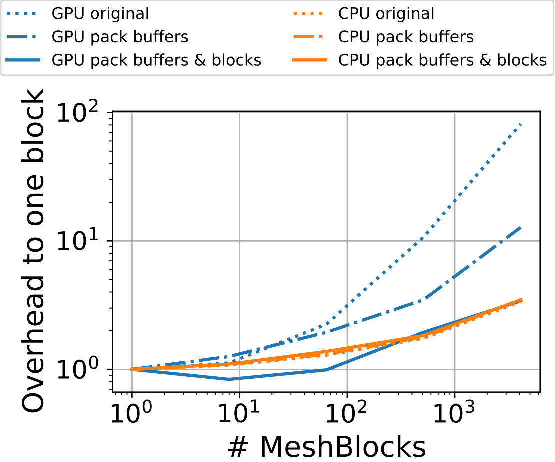
- GPU 256^3 mesh with blocks 256^3 to 16^3
- CPU 128^3 mesh with blocks 128^3 to 8^3

Packing #1: Kernel fusing $\leftrightarrow$ block packing

[Grete+ IJHPCA 2023 – Parthenon collaboration]

- Launch overhead
  - $\approx 5\mu$s launch, inherently serial (launching in parallel does not help)
  - possibly $> 100,000$ buffers per device
- Small blocks $\Rightarrow$ little work
  - $16^3 = 4$k cells $\leftrightarrow$ >1k cores/device
  - even Riemann solve is $< 5\mu$s

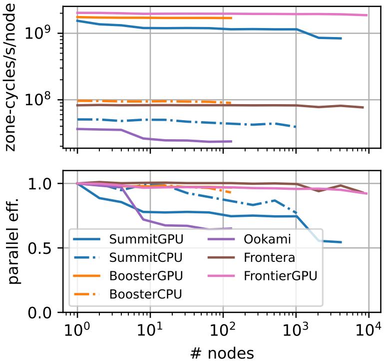
$\Rightarrow$ Combine work into fewer kernels



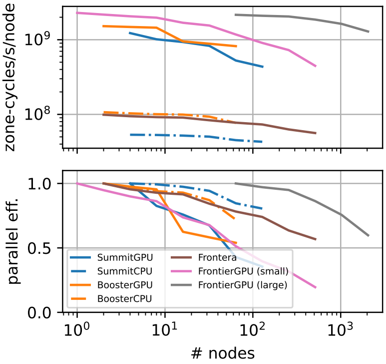
- GPU 256^3 mesh with blocks 256^3 to 16^3
- CPU 128^3 mesh with blocks 128^3 to 8^3

Scaling on TOP500 #1 Frontier

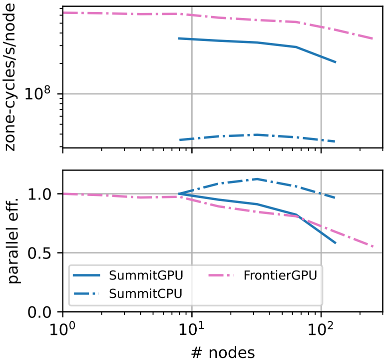
[Grete+ IJHPCA 2023 – Parthenon collaboration]



Uniform mesh (weak)



Uniform mesh (strong)



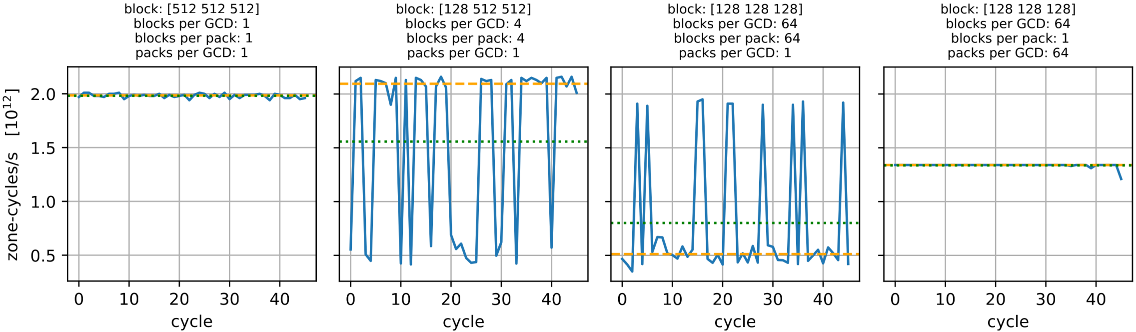
Multilevel mesh (strong)
(24k 32^3 blocks)

$\Rightarrow$ 92% weak scaling efficiency on 73,728 GPUs and

$\Rightarrow$ $\gtrsim$ 50% strong scaling efficiency for 100x increase in resources

Packing #2: Messages in a bottle(neck)

[Holmen, Grete & Melesse Vergara CUG23]



- 1024 nodes
- 16384x8192^2 mesh
- Vary block sizes and pack sizes

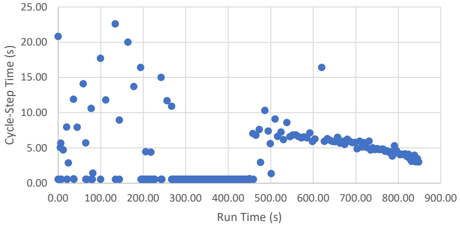
$\Rightarrow$ Messaging matters

$\Rightarrow$ Room for more optimizations

IO #1: 1x9000 vs 9000x1 – What could possibly go wrong?

[Holmen, Grete & Melesse Vergara CUG23]

- `parthenon-hydro` part of OLCF test harness (used for system testing)
- "All nodes" versus "every node" tests
- Goal: Isolate "bad" nodes
- Observed strong variability



performance over time of a random node in "every node" test case

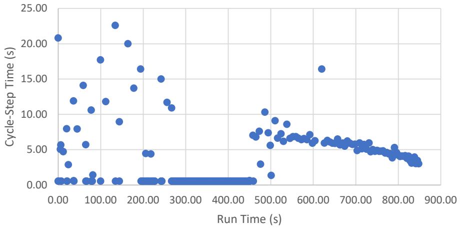
IO #1: 1x9000 vs 9000x1 – What could possibly go wrong?

[Holmen, Grete & Melesse Vergara CUG23]

- `parthenon-hydro` part of OLCF test harness (used for system testing)
- "All nodes" versus "every node" tests
- Goal: Isolate "bad" nodes
- Observed strong variability

⇒ `stat` on parallel file systems does **not** scale

- Potentially relevant to parameter sweeps



performance over time of a random node in "every node" test case

IO #2: Writing (a) “large” file(s)

- Parallel HDF5 (with MPI IO)
- Single file per output
- No issues on Alpine (GPFS)
  - writing 6TB files in $<15\,\mathrm{s}$
  - using collective buffering (one rank per node with 16MB buffer size)

IO #2: Writing (a) “large” file(s)

- Parallel HDF5 (with MPI IO)
- Single file per output
- No issues on Alpine (GPFS)
  - writing 6TB files in $<$15 s
  - using collective buffering (one rank per node with 16MB buffer size)

- On Orion (Lustre) for INCITE runs (2.7TB)
  - 284 s: defaults
  - 25 s: explicit striping*
  - 16 s: explicit striping, no collective buffering
- BUT Lots of (silent) “I/O Errors”

IO #2: Writing (a) “large” file(s)

- Parallel HDF5 (with MPI IO)
- Single file per output
- No issues on Alpine (GPFS)
  - writing 6TB files in <15 s
  - using collective buffering (one rank per node with 16MB buffer size)
- On Orion (Lustre) for INCITE runs (2.7TB)
  - 284 s: defaults
  - 25 s: explicit striping*
  - 16 s: explicit striping, no collective buffering
- BUT Lots of (silent) “I/O Errors”

⇒ Monitoring script for silent failures

- Single file per output does not scale (for us on Lustre)

⇒ HDF5 subfiling (I did not get it working)

⇒ OpenPMD/ADIOS2 (tests successfully wrote 4.5TB file in <1 s)

- * use “capacity” tier not “performance”

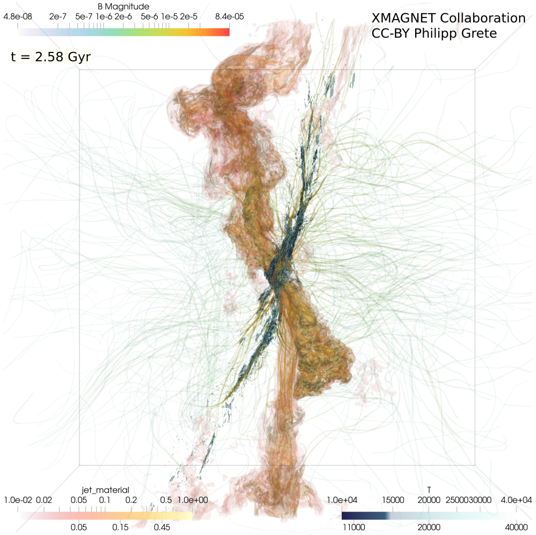
Large scale visualization

Paraview on Andes

- Establishing connection takes a long time (⇒ increase timeout)
- Preselect data (⇒ reduce memory footprint)
- Be patient!

Next: In-situ with Ascent

- Still fighting performance degradation



Conclusions – Take home message(s)

[Grete+ IJHPCA 2023 – Parthenon collaboration]

- Fuse kernels
- Remain flexible wrt. communication
- Do not write to a single large file
- Introduce safety checks (e.g., for timeouts)
- BENCHMARK!

We are an open, welcoming community. Meet us at/on

- `https://github.com/parthenon-hpc-lab`
- Matrix chat: `#parthenon-general:matrix.org`