Parthenon – a performance portable block-structured adaptive mesh refinement framework

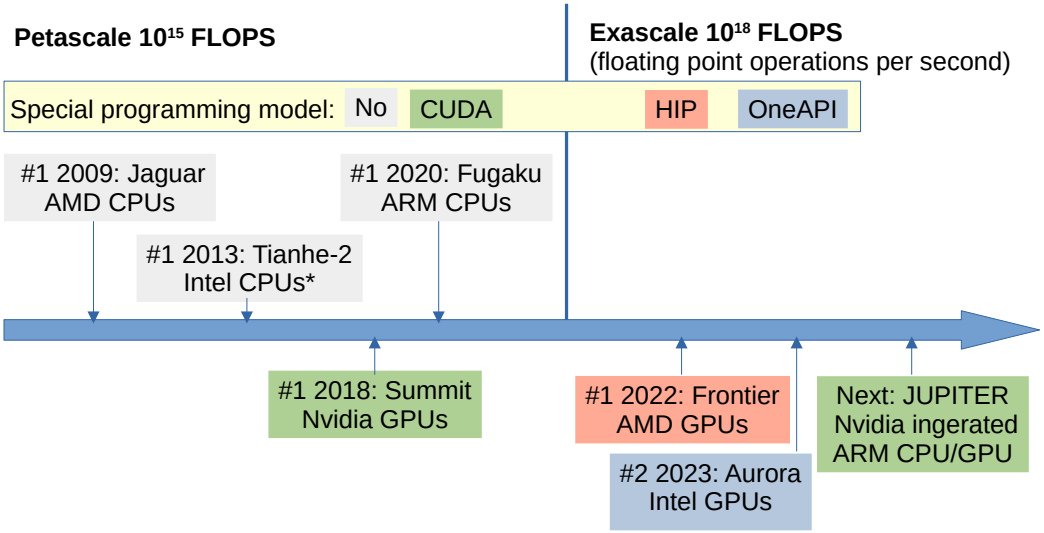
Philipp Grete
Hamburg Observatory

in collaboration with F. Glines, B. O'Shea, J. Holmen, and the Parthenon community (A. Dempsey, J. Dolence, J. Miller, P. Mullen, C. Prather, A. Reyes, L. Roberts, and more)

Kokkos Tea-Time

Funded by
DFG Deutsche Forschungsgemeinschaft
German Research Foundation

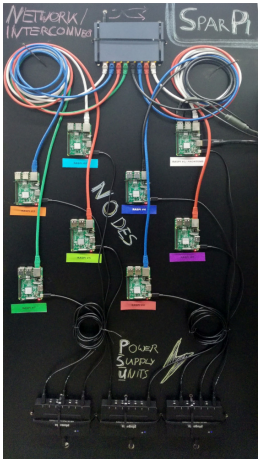
A changing HPC landscape – architecture & programming models



A changing HPC landscape – key challenges

- Computational
  - Multicore ⇒ manycore
  - Diverse hardware **and** programming models
  - Traditional scale-up/out not applicable
- Scientific: Porting code is not sustainable
- Societal: HPC uses large amounts of resources

⇒ Need: effective and efficient use of capabilities



SparPi supercomputer model

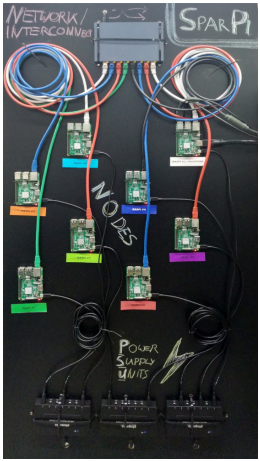
A changing HPC landscape – key challenges

- Computational
  - Multicore ⇒ manycore
  - Diverse hardware **and** programming models
  - Traditional scale-up/out not applicable
- Scientific: Porting code is not sustainable
- Societal: HPC uses large amounts of resources

⇒ Need: effective and efficient use of capabilities

Performance portability... *broadly refers to portable (single) source code that makes efficient use of the hardware architecture whatever it may be*

[performanceportability.org]



SparPi supercomputer model

From CPU to any GPU

[Grete, Glines, & O'Shea TPDS 2021]
[Grete+ IJHPCA 2022]

Standard C++ CPU code

- Example 3D loop:

```
for( int k = ks; k < ke; k++){
 for( int j = js; j < je; j++){
  #pragma omp simd
  for( int i = is; i < ie; i++){
   /* Loop Body */
   u(k,j,i) = ...
}}}
```

From CPU to any GPU

[Grete, Glines, & O'Shea TPDS 2021]
[Grete+ IJHPCA 2022]

Standard C++ CPU code

- Example 3D loop:

```
for( int k = ks; k < ke; k++){
 for( int j = js; j < je; j++){
  #pragma omp simd
  for( int i = is; i < ie; i++){
   /* Loop Body */
   u(k,j,i) = ...
}}}
```

Kokkos abstraction (exposes full control)

```
parallel_for(team_policy(nk, AUTO),
 KOKKOS_LAMBDA(member_type team) {
  const int k = team.league_rank()+ks;
  parallel_for(
   TeamThreadRange<>(team,js,je,
    [&] (const int j) {
     parallel_for(
      ThreadVectorRange<>(team,is,ie,
       [=] (const int i) {
        /* Loop Body */
        u(k,j,i) = ...
});});});});
```

From CPU to any GPU

[Grete, Glines, & O'Shea TPDS 2021]
[Grete+ IJHPCA 2022]

Standard C++ CPU code

- Example 3D loop:

```
for( int k = ks; k < ke; k++){
 for( int j = js; j < je; j++){
  #pragma omp simd
  for( int i = is; i < ie; i++){
   /* Loop Body */
   u(k,j,i) = ...
}}}
```

Our (and other) codes

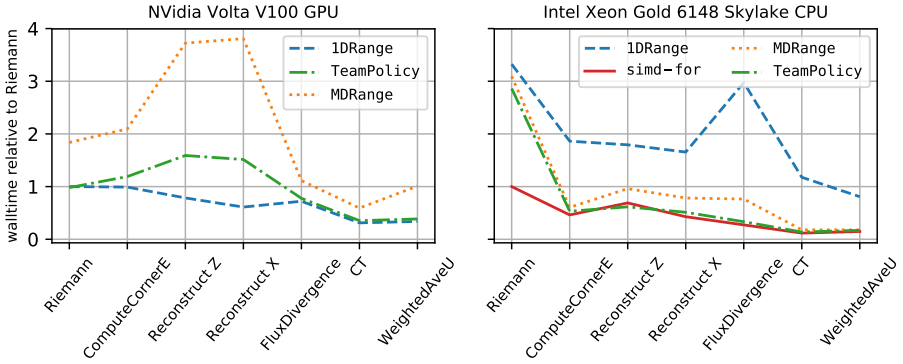
```
par_for("name",ks,ke,js,je,is,ie,
 KOKKOS_LAMBDA (int k,int j,int i) {
  /* Loop Body */
  u(k,j,i) = ...
});
```

- Introduced intermediate abstraction layer
- Runs on any GPU (and CPU)
- Simple things stay simple
- No expert knowledge required

⇒ proof-of-concept code `K-Athena` with 76% parallel efficiency on all of Summit

Intermediate abstraction layer and execution policies

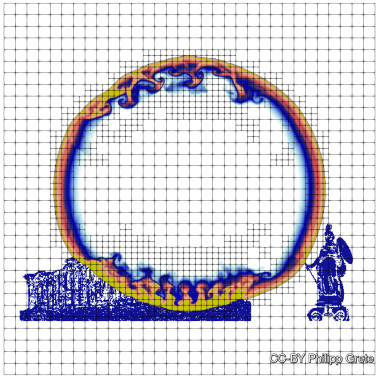
[Grete, Glines, & O'Shea TPDS 2021]



- Execution policies have significant impact on performance
- No "one size fits all" solution yet
- In some codes: manual pointers extraction to aid compiler vectorization

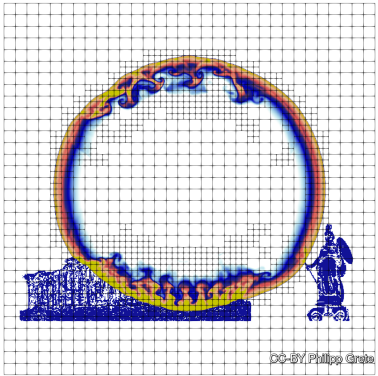
(Adaptive) Mesh Refinement (AMR)

- 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



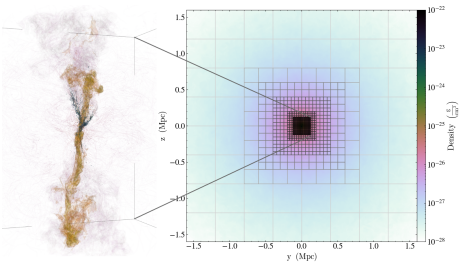
(Adaptive) Mesh Refinement (AMR)

- 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



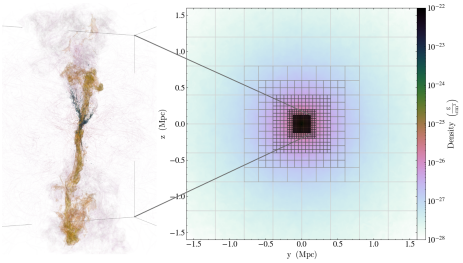
Parthenon – Performance portable AMR framework

- Goal: Hide complexity of device-accelerated AMR
- Open collaboration (20+ contributors)
  github.com/parthenon-hpc-lab
- Key performance design decisions
  - device first/resident
  - block packing/kernel fusing
  - device-to-device communication via async. MPI



Parthenon – Performance portable AMR framework

- Goal: Hide complexity of device-accelerated AMR
- Open collaboration (20+ contributors) github.com/parthenon-hpc-lab
- Key performance design decisions
  - device first/resident
  - block packing/kernel fusing
  - device-to-device communication via async. MPI
- Advanced features (e.g., abstract data containers, package system, task-based parallelism, sparse variables, particles, integrated solvers)
- Multiple downstream codes

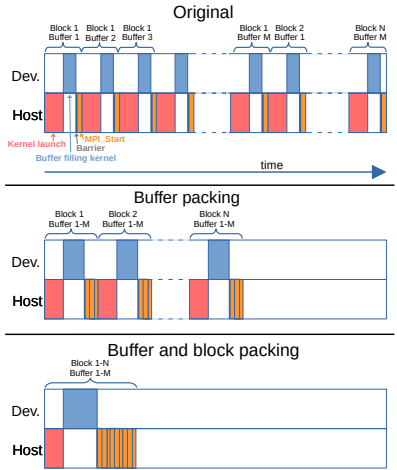


Packing: Kernel fusing ↔ block packing

[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 $\leftrightarrow$ >1k cores/device
  - even Riemann solve is $< 5\mu$s

$\Rightarrow$ Combine work into fewer kernels

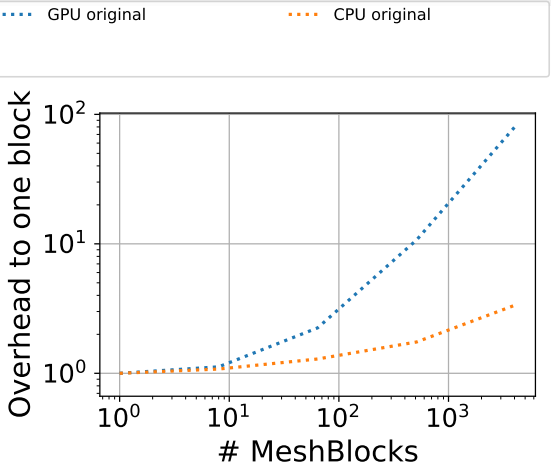


Packing: Kernel fusing ↔ block packing

[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 ⇒ little work
  - $16^3 = 4$k cells ↔ >1k cores/device
  - even Riemann solve is $< 5\mu$s

⇒ Combine work into fewer kernels



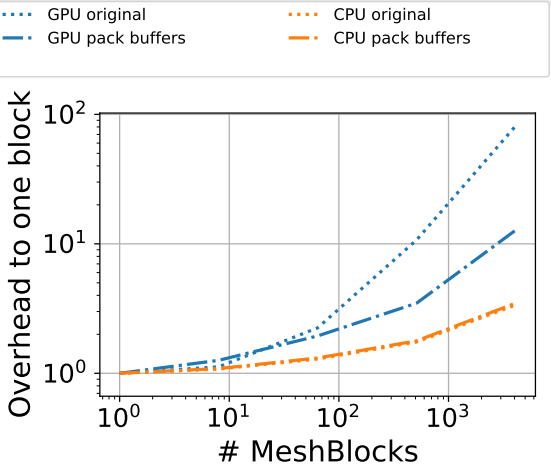
- GPU 256^3 mesh with blocks 256^3 to 16^3
- CPU 128^3 mesh with blocks 128^3 to 8^3

Packing: 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 ⇒ little work
  - $16^3 = 4$k cells ↔ >1k cores/device
  - even Riemann solve is $< 5\mu$s

⇒ Combine work into fewer kernels



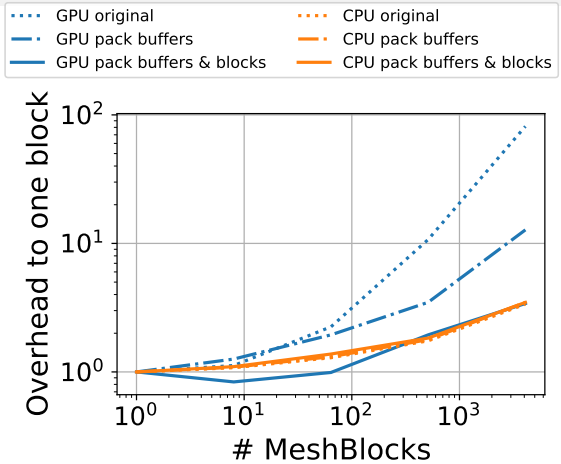
- GPU 256^3 mesh with blocks 256^3 to 16^3
- CPU 128^3 mesh with blocks 128^3 to 8^3

Packing: 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 ⇒ little work
  - $16^3 = 4$k cells ↔ >1k cores/device
  - even Riemann solve is $< 5\mu$s

⇒ Combine work into fewer kernels

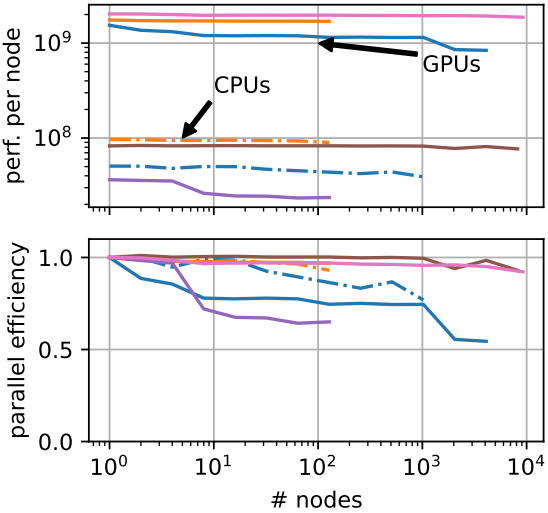


- GPU 256^3 mesh with blocks 256^3 to 16^3
- CPU 128^3 mesh with blocks 128^3 to 8^3

Parthenon – a performance portable AMR *framework*

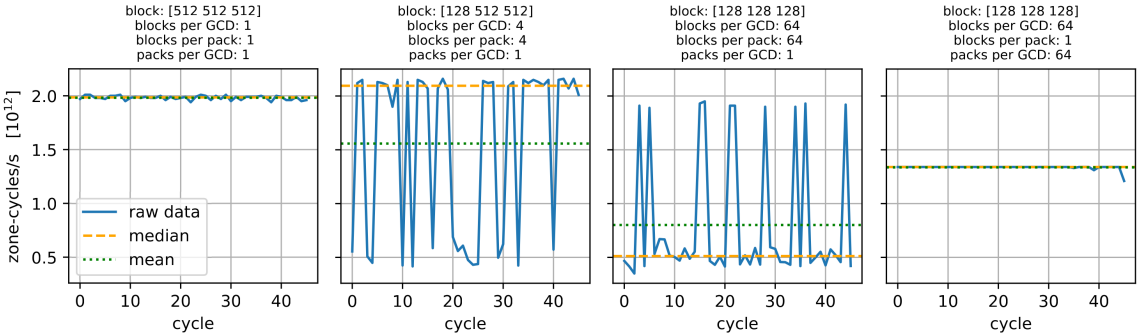
[Grete+ IJHPCA 2022]

- Performance across architectures:
  - CPU: x86, ARM, Power9
  - GPU: CUDA, HIP
- #1 TOP500: Frontier (9472 nodes)
  - 92% weak scaling efficiency on 73,728 GPUs (eff. 500 million cores)
- Performance portable GPU-acceleated AMR works in practice*!



Packing #2: Messages in a bottle(neck)

[Holmen, Grete & Melesse Vergara CUG23]



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

$\Rightarrow$ Messaging matters

$\Rightarrow$ Support for coalesced messages with index splitting ($\rightarrow$ incr. occupancy)

Advanced features: packing of (sparse) variables on blocks

- `View` + `Metadata` = `Variable`
- Multiple `Variable` = `MeshBlockData`
- `MeshBlockData` across multiple blocks = `MeshData`
- Arb. selection of variable and blocks = `Pack` (read: `View` of `View` (of `View`)

```
struct my_var_t : public variable_names::base_t<false> {
  template <class... Ts>
  KOKKOS_INLINE_FUNCTION varname(Ts &&...args)
      : variable_names::base_t<false>(std::forward<Ts>(args)...) {}
  static std::string name() { return "my_var"; }
}
...

pkg->AddField<my_var_t>(Metadata(...));
pkg->AddSparsePool<my_sparse_var_t>(Metadata(...), sparse_idxs);
...
```

Define a variable name type that uniquely identifies your field. Needs to inherit from `variable_names::base_t` and provide `name()` which returns a unique string.

Add fields and sparse fields to your package using the type

[Credit: L. Roberts LA-UR 25-25420]

Advanced features: packing of (sparse) variables on blocks cont.

```
/*static*/ auto desc = MakePackDescriptor<my_var_t, my_other_var_t>(md);
...

auto pack = desc.GetPack(md);
...

par_for(PARTHENON_AUTO_LABEL, 0, pack.GetNBlocks() - 1, kb.s, kb.e, jb.s, jb.e, ib.s, ib.e,
    KOKKOS_LAMBDA(const int b, const int k, const int j, const int i) {
      pack(b, TE::F1, my_var_t(), k, j, i) = ...;
      for (int l = pack.GetLowerBound(b, my_sparse_var_t());
           l <= pack.GetUpperBound(b, my_sparse_var_t()); ++l) {
        pack(b, my_sparse_var_t(l), k, j, i) = ...; // Default assumption is TE::CC
      }
    });
```

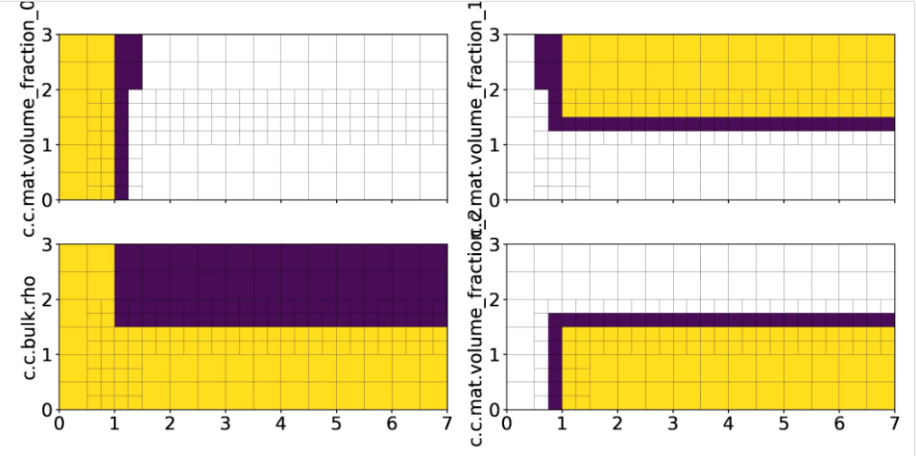
A `PackDescriptor` defines a list of fields to be packed. Also can take `PDOpt` options and `Metadata` flags to further define selection of fields.

Actually get the pack from a `MeshData` object

Access the pack in a kernel

[Credit: L. Roberts LA-UR 25-25420]

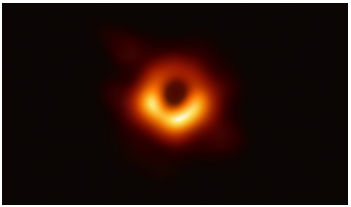
Sparse variables in action (white blocks are not allocated)



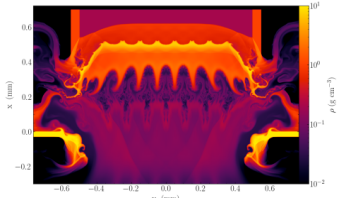
[Credit: J. Dolence LA-UR-26-22714]

Parthenon-enabled applications and ecosystem

- `AthenaPK` (astrophysical) MHD and turbulence
- `Phoebus` GRRMHD (e.g., neutron star mergers)
- `KHARMA` GRMHD (used within EHT collaboration)
- `RIOT` multi-material physics
- `Artemis` multi-fluid, radiation hydro
- `Flash` fusion power
- `parthenon-hydro`: just 1000 lines of code including test infrastructure ($\Rightarrow$ tutorials/teaching)



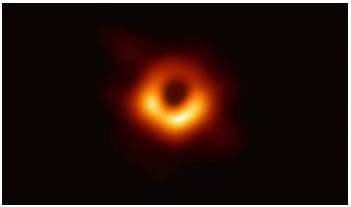
[EHT collaboration]



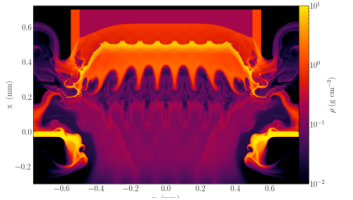
[RIOT LA-UR-26-22714]

Parthenon-enabled applications and ecosystem

- `AthenaPK` (astrophysical) MHD and turbulence
- `Phoebus` GRRMHD (e.g., neutron star mergers)
- `KHARMA` GRMHD (used within EHT collaboration)
- `RIOT` multi-material physics
- `Artemis` multi-fluid, radiation hydro
- `Flash` fusion power
- `parthenon-hydro`: just 1000 lines of code including test infrastructure ($\Rightarrow$ tutorials/teaching)
- `Singularity-eos` microphysics
- `Jaybenne` Monte Carlo radiation transport
- ...



[EHT collaboration]



[RIOT LA-UR-26-22714]

Pushing limits with AthenaPK through Parthenon

- Own downstream code: AthenaPK targeting processes/systems involving astrophysical plasmas
  - Supernova remnants
  - Cloud/galaxies in wind (example right)
  - Jets from active galaxies
  - Magnetized turbulence
    GCS Exascale Pioneer Application on JUPITER

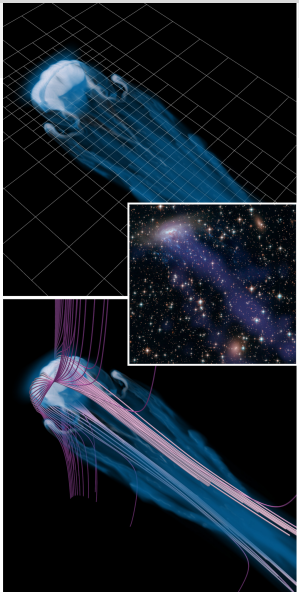
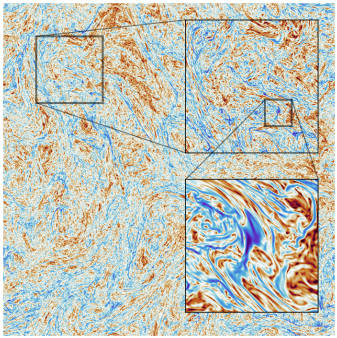


Image credit (inset): NASA/HST

Pushing limits with AthenaPK through Parthenon

- Own downstream code: AthenaPK targeting processes/systems involving astrophysical plasmas
  - Supernova remnants
  - Cloud/galaxies in wind
  - Jets from active galaxies
  - Magnetized turbulence
    GCS Exascale Pioneer Application on JUPITER
- Successfully (more or less)
  - running 4 trillion cells MHD turbulence simulation
  - writing/restarting from 289TB checkpoint
  - running with 26 cell wide ghost layer
  - running with particles 57 fields



MHD turbulence on JUPITER

Conclusions and outlook

- Performance portable AMR works in practice*
  (*but needs non-trivial pooling of resources)
- Enabled by `Kokkos`
- Large, collaborative effort!
- Current challenges
  - High register pressure kernels
  - Memory pools
  - Efficient vectorization
- We are an open, welcoming community. Meet us at/on
  - `https://github.com/parthenon-hpc-lab`
  - Matrix chat: `#parthenon-general:matrix.org`