RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

cea cnrs Inria

FRANCE 2030 PROGRAMME DE RECHERCHE NUMÉRIQUE POUR L'EXASCALE

Exa-DI - WG3
Block-structured Adaptive Mesh Refinement

Maxime Delorme (CEA)

Applicative teams: Dyablo (CEA), Samurai (Polytechnique)

Adaptive Mesh Refinement

How to simulate orders of magnitude in spatial and temporal scales ?

AMR solves the problem of multi-scale *sparse* computation

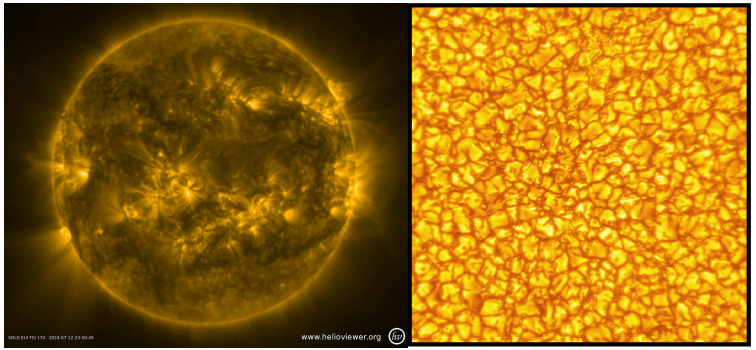
- *i.e.* Very large domains, with fine details localized in some regions
- Allows to push the resolution much farther then regular grid
- At the cost of algorithmic complexity, in particular difficult to manipulate data-structures.

AMR comes in multiple flavors

- Type of AMR: Cell-based, block-based, patch-based
- Data structure: *Interval-based, tree-based, linear tree-based*, unstructured

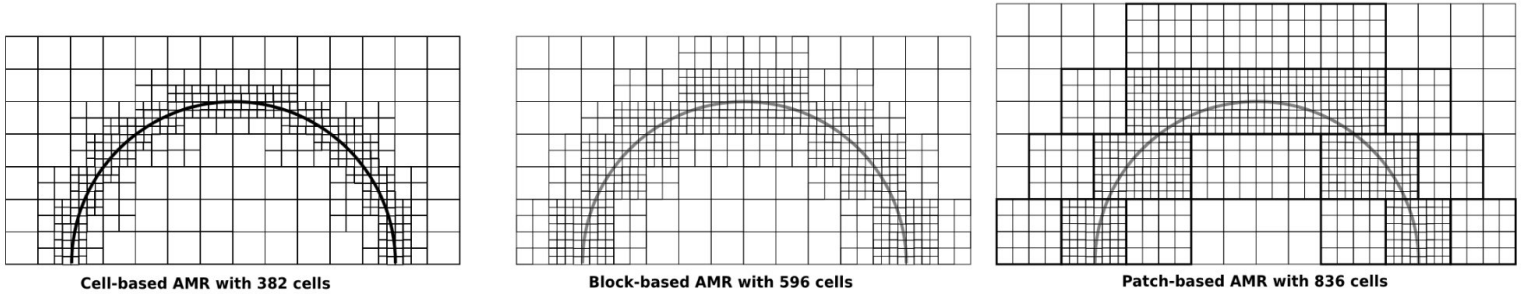
AMR is equation independent

- Can cater to any sort of problem that requires a grid



The Sun is ~2M km wide, solar granulation is ~1000km wide
Sources:
Left: www.helioviewer.org
Right: NSF/AURA/NSO Image Processing: Friedrich Wöger(NSO), Catherine Fischer (NSO)

Block Structured Adaptive Mesh Refinement



Source: Dunning et al. 2019; "Adaptive Mesh Refinement in the Fast Lane."

Block structured:

- Tree or interval is possible
- Each "unit" of the AMR structure is a regular cartesian block (eg 4x4, 8x8, 19x31)

Why use blocks ?

- **Pro:** Shallower structures with less levels
- **Con:** Makes refinement "chunkier"
- However blocks
  - Increases regularity of structures on the GPU
  - Shallower data structures are easier to balance and distribute

Exa-DI WG3

What is the question ?

Various formalisms for block-structured AMR lead to different problems and advantages

- Some problems might be independent of flavor and common to all codes
- Main questions:
  - What are the main common bottleneck that most AMR codes will face on exascale machines ?
  - Can we find a common solution to these problems ?
  - How can the rest of Numpex help us ?
- Examples of anticipated problems:
  - Efficient IOs and AMR data format
  - Load balancing
  - Large global operations (eg linear solvers)
  - General scaling and performance assessment

Current state

Codes

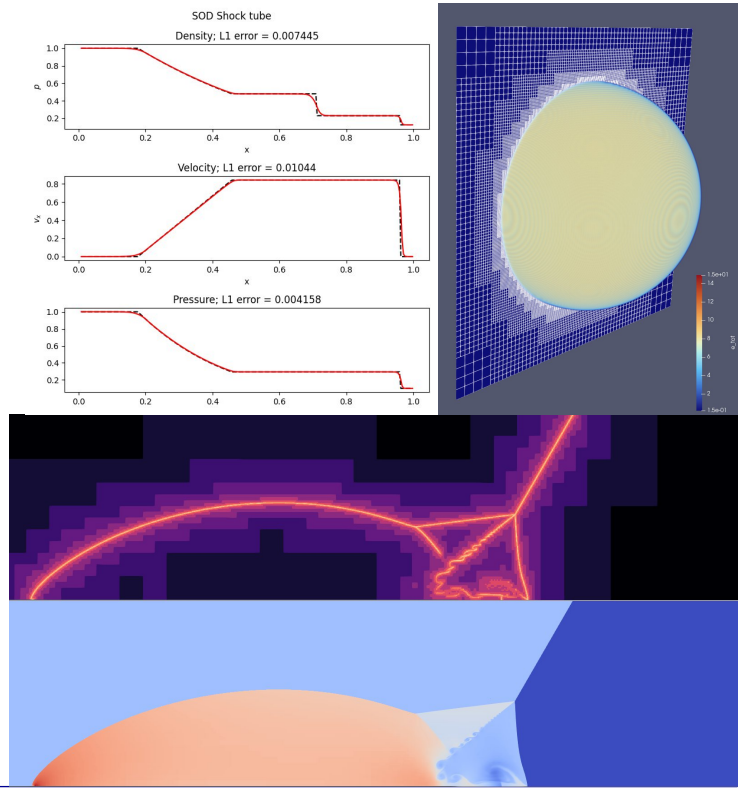
- Samurai (Polytechnique/CMAP)
- Dyablo (CEA/IRFU)

Benchmark problems

- Three key problems defined around the Euler equations
  - 1. Non-axis aligned Sod shock tube
  - 2. Sedov Blast
  - 3. Double Mach Reflection

Deployment

- GUIX Packaging for Dyablo
- Pipeline development
  - Automatic launching of benchmarks
  - Parameter space exploration
  - Post-treatment and analysis of results



Roadmap

Jan

- Finalizing benchmark tests
- ~~Defining metrics for benchmark evaluation~~
- ~~Dynamic access allocation on GENCI systems~~
- Finalizing pipelines
- ~~First runs on Grid5000~~

Feb-Mar

- First runs on JZ and AA
- Full analysis reporting on gitlab
- Identification of bottlenecks on CPU for Samurai and Dyablo
- Identification of main GPU bottlenecks for Dyablo

Apr

- Integration of Athena-K to the benchmark
- Integration of Castro to the benchmark

CDT Application teams Both

Roadmap

Pipelines

- Current pipeline is being switched with IOPS for automated benchmarks on any supercomputer

Benchmark tests

- Sedov blast benchmark is missing an analytical profile for quality of solution calculation
- Dyablo has a bug when running the Double Mach at certain resolutions

Additional codes

- **Castro**: Based on AMRex, flagship patch-based AMR framework for ECP, good baseline comparison
- **AthenaK**: Athena++ porting to Kokkos, block-based AMR
- *Kalypsso: Block-based AMR based on Kokkos from CEA-DAM, very similar to Dyablo*

Metrics for an AMR performance benchmark

Literature always measures in some flavor of MCell-update/s

- But is that really important ?
- An implicit AMR code can do much more than a regular-grid explicit code, but will update less cells/s !
- Result will also depend on the topology of the AMR structure.
  - For a given number of cells, deeply nested structures will give different results than shallow trees.

What is important ?

- For a given problem, performance will be assessed by :
  - The quality of the solution -> Quality metric (L1, L2 wrt a reference solution)
  - The efficiency to get this solution -> Time to solution, Watt to solution
- Quality of solution will depend on the resolution but also the solvers
  - For starters we keep the solver identical between codes (2nr order in time and space; SSP-RK2 + MUSCL) and explore resolution effects
  - We compute the L2 norm wrt a reference solution: Either analytical (Sod, Sedov blast) or very high resolution of the run in the code (Double Mach) -> Easy to realize
  - We also compute the time-to-solution for a given solution. We then get for each run a L2-norm vs time-to-solution.

Perspectives

Bottleneck identification

- What are the common bottlenecks AMR codes have to face for exascale level productions ?
- Are these bottlenecks the same on CPU and GPU ? On Nvidia and AMD ?
- What solutions proposed by the rest of the PCs of Numpex can solve these bottlenecks ?
  - Spoiler: A first work in on-going for Dyablo and Samurai in PC3 (EXADost)
- Can some of these bottlenecks be solved by a direct collaboration with AMD/Bull ?
- What can be learned from similar codes abroad (eg Castro and Athena-K) ?