CarpetX

2021 Einstein Toolkit Summer School, UIUC

Erik Schnetter, Perimeter Institute

2021-07-27

Flesh and Thorns

Plug-In "Thorns" (modules)

driver

input/output

interpolation

SOR solver

wave evolvers

multigrid

coordinates

extensible APIs

ANSI C

parameters

scheduling

Core "Flesh"

error handling

make system

grid variables

utilities

boundary conditions

remote steering

Fortran/C/C++

equations of state

Your Physics !!

Your Computational Tools !!

black holes

Tom Goodale

Cactus Tutorial
2007-10-17

The Cactus framework

- The **Driver** thorn in Cactus is the “main” function of the simulation:

- Scheduling (calls physics functions)
- Parallelism (splits grid functions across processes, exchanges ghost zones)
- AMR (levels, boxes, interpolation) (**AMReX!**)
- I/O (reading, writing, checkpointing)
- …

Outline

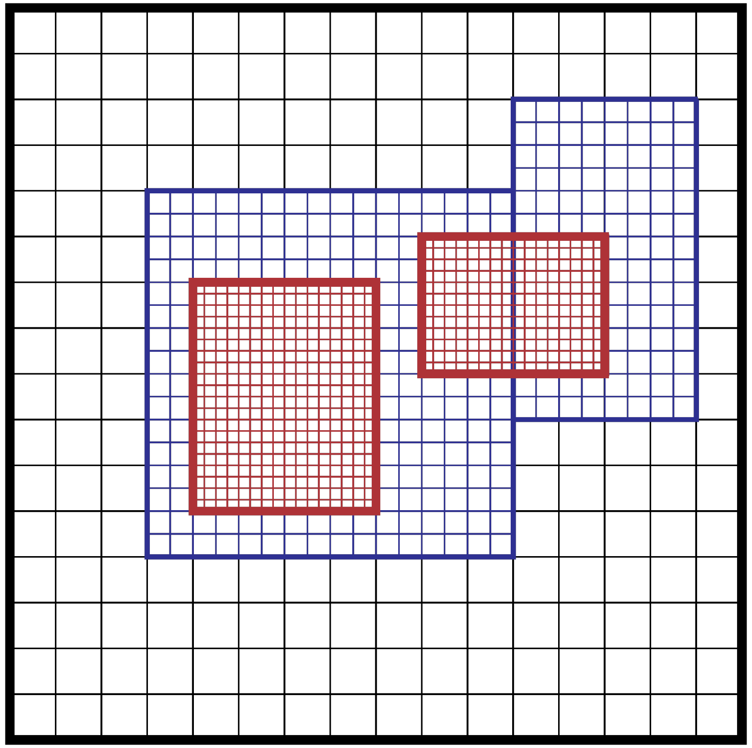
- Tasks of a driver thorn
- Why AMR (Adaptive Mesh Refinement)
- CarpetX: New features (and missing features)

- New safety features
- Vertex and cell centering (conservation laws, div B)
- Parallelisation and efficiency (MPI, OpenMP, GPUs)

- I/O, SIMD, …

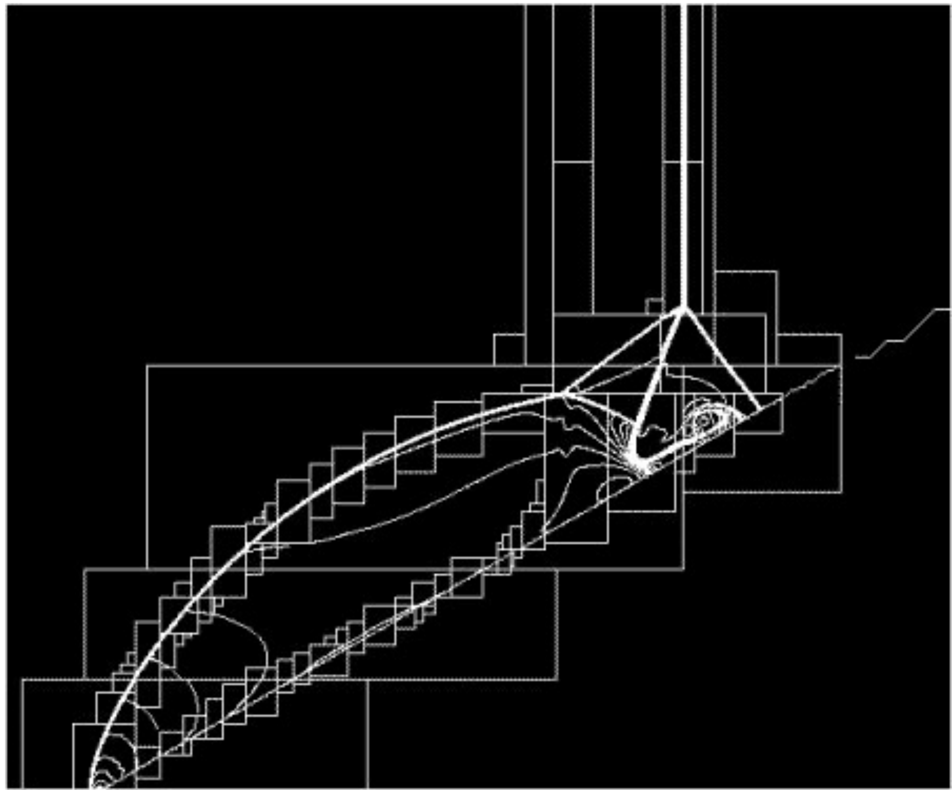
Adaptive Mesh Refinement

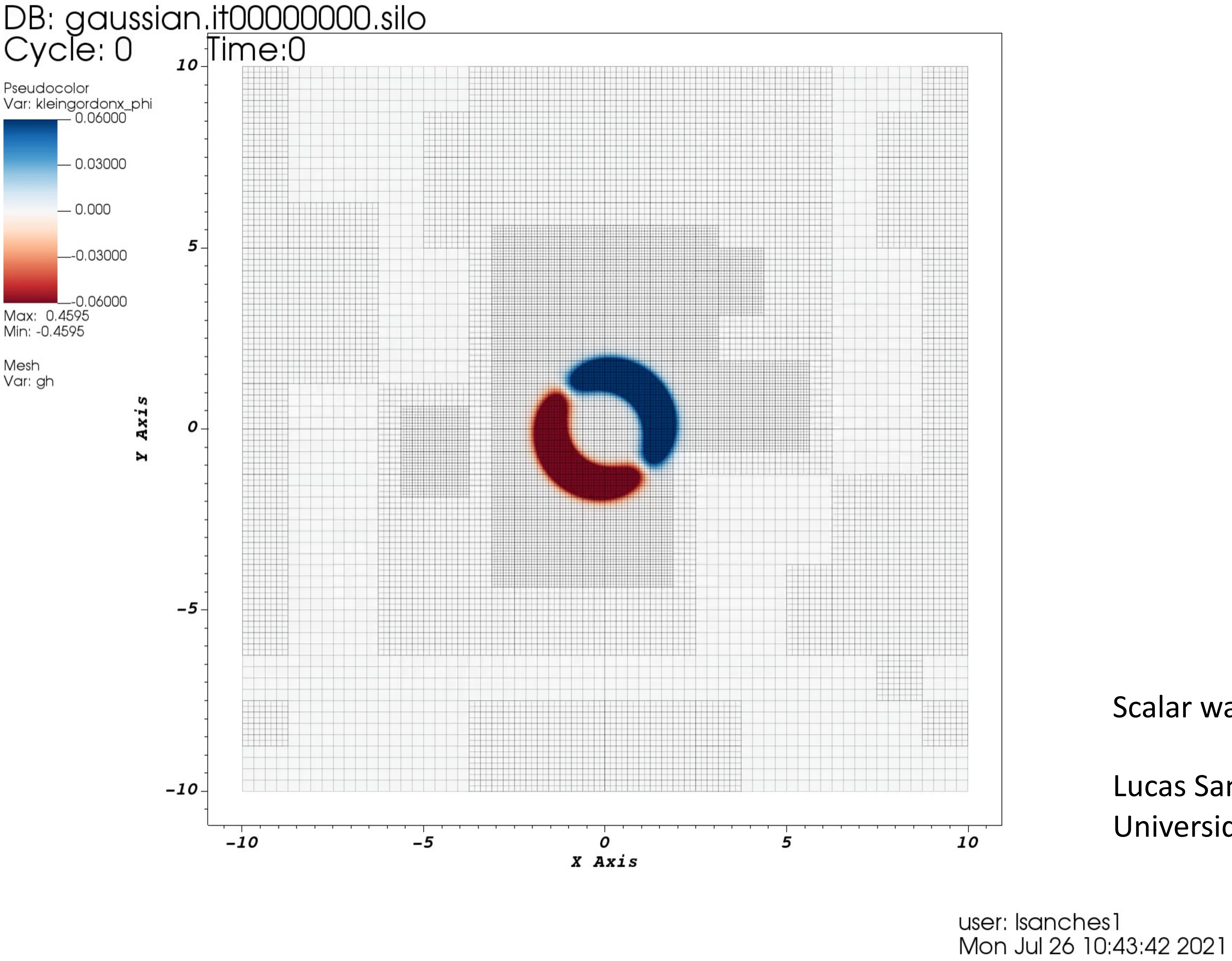
- Need very high resolution only in small part of domain
- Often used for **compressible hydrodynamics**:
  - Moving shocks, discontinuities, surfaces



[AMReX documentation]

[Wikipedia: AMR]

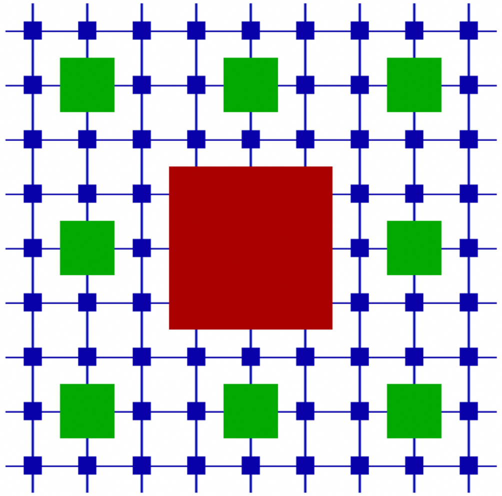




Scalar wave on Minkowski background

Lucas Sanches
Universidade Federal do ABC (Brazil)

CarpetX – a new mesh refinement driver for numeric relativity



- Designed for the Einstein Toolkit
- True adaptive mesh refinement based on local error estimate
- High-order prolongation/ restriction operators
- Vertex/cell/face/edge-centred variables
- Refluxing for exact conservation in (M)HD
- can currently run a qc0 BBH merger
- see presentations by Don Willcox and Erik Schnetter

- Uses AMReX
- for Exascale scalability
- Much improved multi-threading
- Much improved I/O speed (ADIOS2, openPMD)
- Improved SIMD vectorization
- Works with GPUs (CUDA)

[Roland Haas]

CarpetX: Missing features

- Not yet implemented:
  - Global time stepping; no subcycling in time
  - Rotational symmetry boundaries
  - GRMHD code
  - NRPy+ backend

- Currently unknown (untested):
  - Recipes for efficient GPU kernels
  - MPI scalability (but AMReX is scalable)

New Safety Features

Scheduling

- Initial conditions:
  1. CCTK_BASEGRID
  2. Loop:
     1. Initialisation done?
     2. Regrid
        - CCTK_BASEGRID
        - CCTK_POSTREGRID
     3. CCTK_INITIAL
     4. CCTK_ POSTINITIAL
  3. CCTK_POSTSTEP
  4. CCTK_ANALYSIS
  5. OutputGH
- Evolution:
  1. Evolution done?
  2. Regrid
     - CCTK_BASEGRID
     - CCTK_POSTREGRID
  3. CCTK_PRESTEP
  4. CCTK_EVOL
  5. CCTK_POSTSTEP
  6. CCTK_ANALYSIS
  7. OutputGH

Scheduling

- BASEGRID:
  - Set up constant data (e.g. coordinates)
- INITIAL, POSTINITIAL:
  - Initialise state vector (including boundaries)
  - Define local error for regridding
- EVOL:
  - Evolve state vector (and set boundaries)
- POSTSTEP, ANALYSIS:
  - Calculate ephemeral values (e.g. constraints)
  - Define local error for regridding
- POSTREGRID:
  - Re-apply boundary conditions to state vector

schedule.ccl:

```
SCHEDULE GROUP WaveToyCPU_RHSGroup IN ODESolvers_RHS
{
} "Calculate RHS"

SCHEDULE WaveToyCPU_RHS IN WaveToyCPU_RHSGroup
{
  LANG: C
  READS: phi(everywhere) psi(everywhere)
  WRITES: phirhs(interior) psirhs(interior)
} "Calculate RHS for the wave equation"

SCHEDULE WaveToyCPU_RHSSync IN WaveToyCPU_RHSGroup AFTER WaveToyCPU_RHS
{
  LANG: C
  SYNC: rhs
} "Boundary conditions for the RHS of the wave equation"

SCHEDULE WaveToyCPU_RHSBoundaries IN WaveToyCPU_RHSGroup AFTER WaveToyCPU_RHSSync
{
  LANG: C
  WRITES: phirhs(boundary) psirhs(boundary)
} "Boundary conditions for the RHS of the wave equation"
```

Declaring Dependencies in the Schedule

- Each scheduled function needs to declare which parts of which variables it reads or writes
- Regridding, synchronization, prolongation etc. do this implicitly as well
- The driver checks consistency, and flags errors
- The driver cross-checks these declarations via:
  - Undefined variables
  - Const variables
  - Variables set to nan, and checked for nan
  - Checksums of parts of variables
- Also, there are “informative” error messages

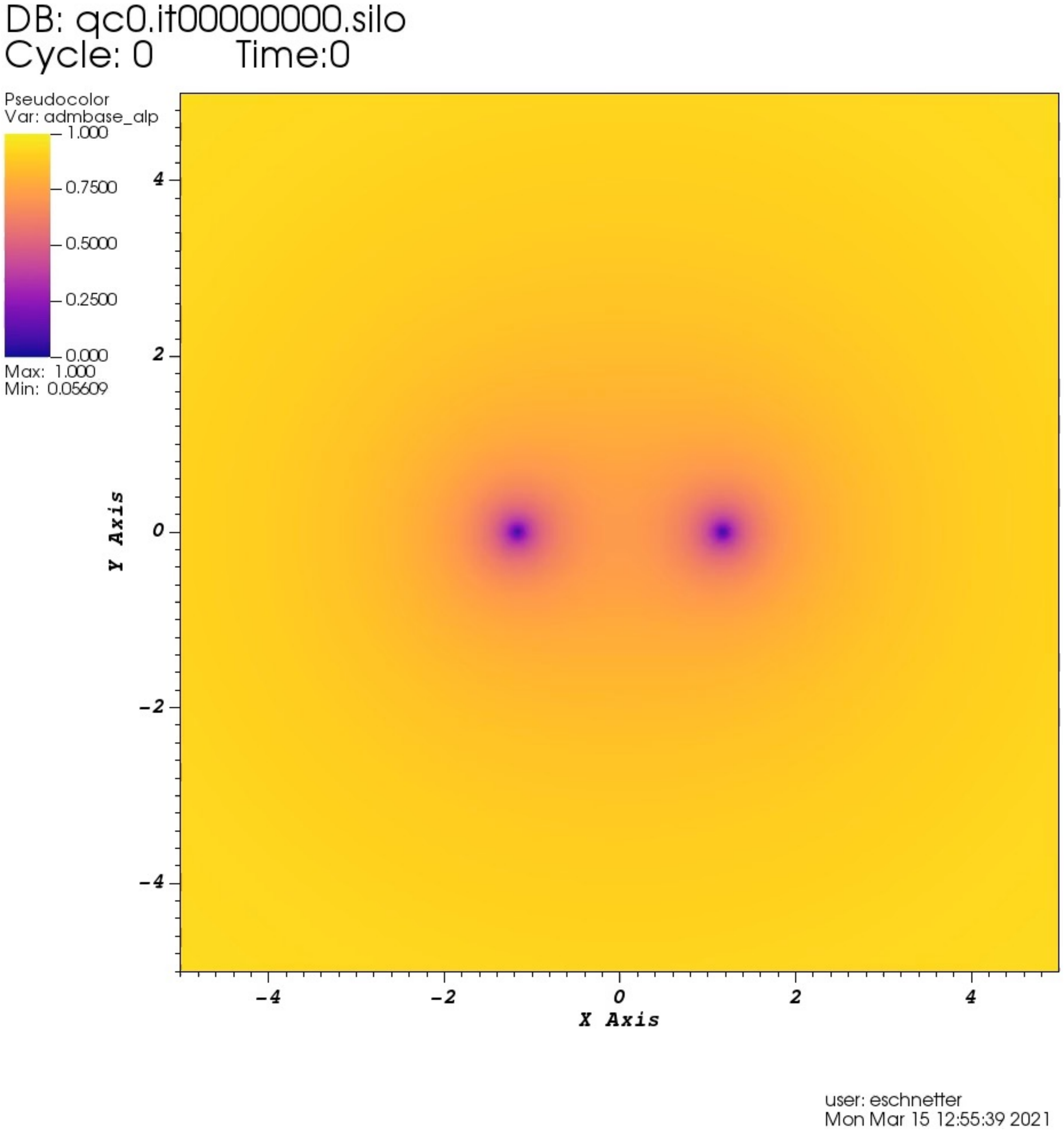
QC-0
(black hole binary)

Large box (+/- 128)

dx[coarse] = 1
dx[fine] = 1/24

4th order everything

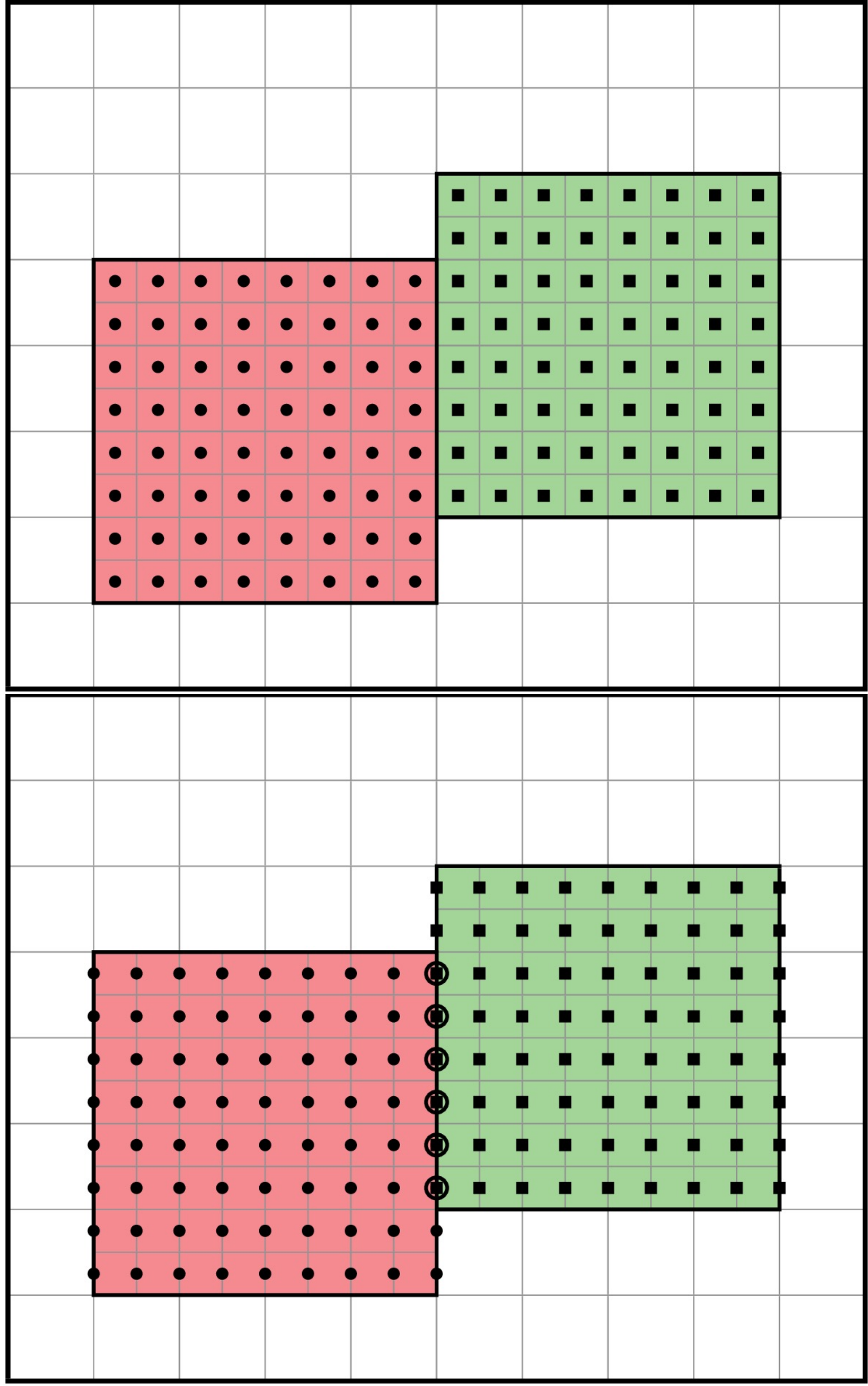
~3/4 orbit before merging



Vertex and Cell centering

Vertex and Cell centering

- ”Points” can be located at vertices, edges, faces, and interior of cells (in 3D)
  - Sample function at vertex
  - Integrate function along edge
  - Average function over cell
- Allows “baking in” certain properties into discretization scheme
  - **topological invariants**
  - E.g.   div B = 0,   baryon number conservation



[AMReX]

interface.ccl:

```
CCTK_REAL state TYPE=gf TAGS='index={1 1 1} rhs="rhs"'
{
  phi psi
} "Scalar potential for wave equation"

CCTK_REAL rhs TYPE=gf TAGS='index={1 1 1} checkpoint="no"'
{
  phirhs psirhs
} "RHS for scalar potential for wave equation"

CCTK_REAL energy TYPE=gf TAGS='index={1 1 1} checkpoint="no"'
{
  eps
} "Energy density for wave equation"

CCTK_REAL err TYPE=gf TAGS='index={1 1 1} checkpoint="no"'
{
  phierr psierr
} "Error of wave equation solution"
```

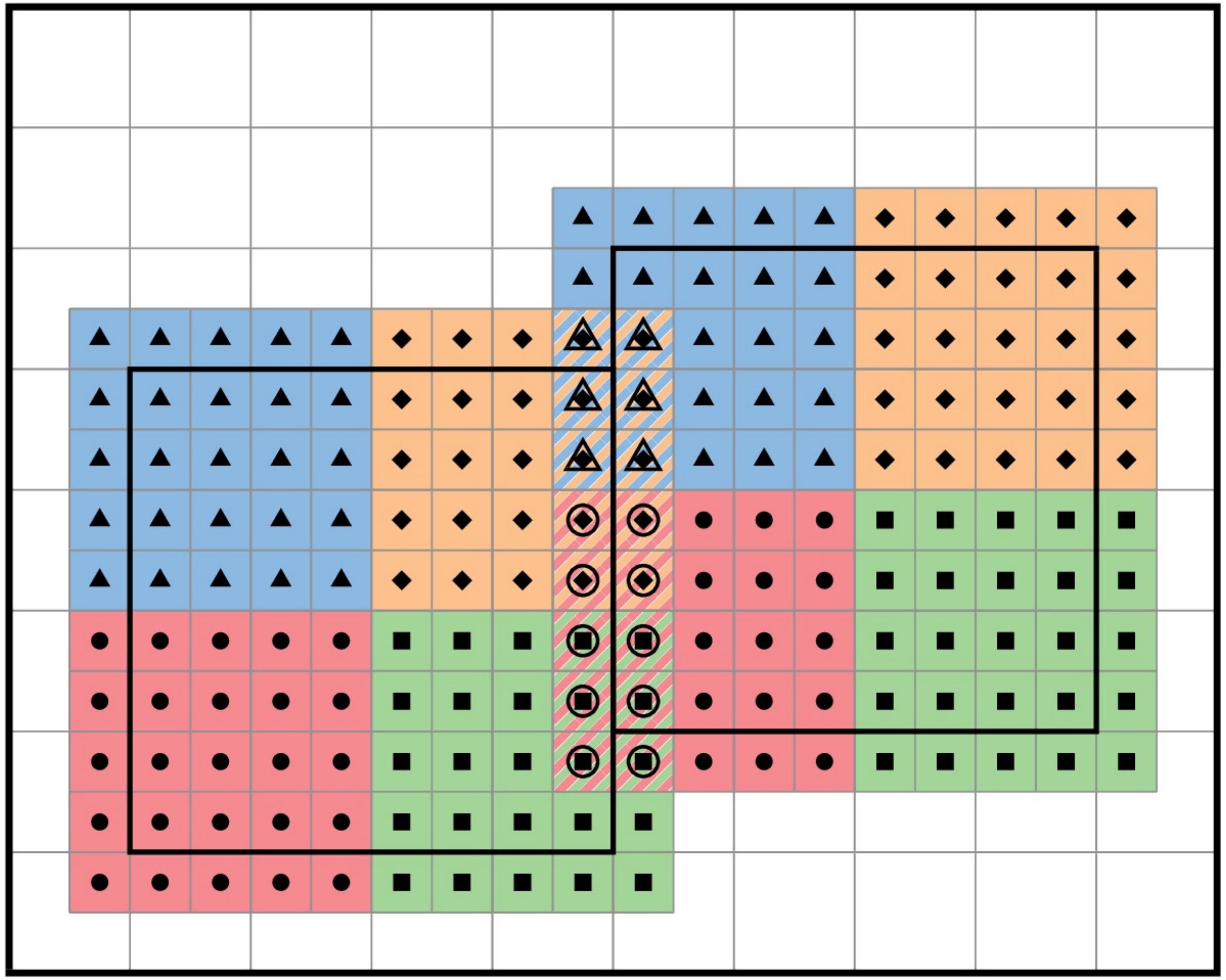
Parallelisation and Efficiency

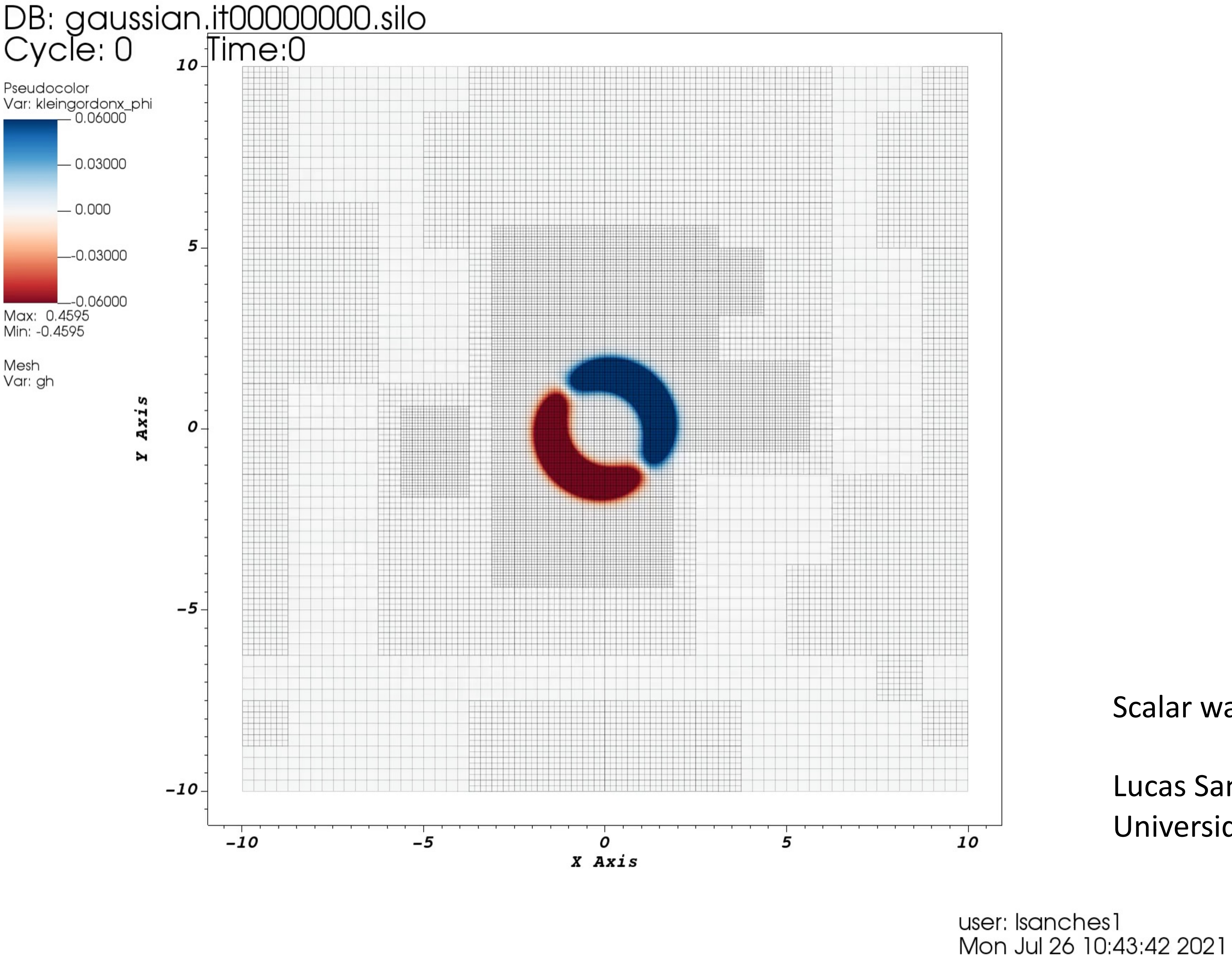
Parallelisation and Efficiency

- **Shared** memory: Many cores can access the same memory
  - Need to split loops, but can use same array
  - Easy to use, but only for small problems (workstation)
  - Multi-**threading** (OpenMP)
- **Distributed** memory: No common memory
  - Need to split data structures as well as loops
  - Difficult but necessary (HPC systems)
  - Separate **processes** (MPI)
- **Accelerators** (“GPUs”):
  - Each accelerator is a shared memory system
  - HPC systems have many accelerators, i.e. distributed memory

Ghost Zones

- **Tiling** for shared memory
  - Each block has 4 coloured tiles
- **Ghost zones** for distributed memory
  - Each block is surrounded by a layer of extra points
  - Ghost zones need to be kept consistent (**synchronization**)
- AMR:
  - **Prolongation** (interpolation from coarse to finer level ghosts)
  - **Restriction** (from fine to coarser level where they overlap)





Scalar wave on Minkowski background

Lucas Sanches
Universidade Federal do ABC (Brazil)

Miscellaneous

I/O

- Need standardized file formats and standardized metadata
  - For post-processing, visualization, initial data exchange, etc.
- (Low-level) file formats can store blocks of data
  - HDF5, ASDF, ADIOS2, …
- (High-level) metadata describes how blocks need to be assembled
  - Silo, openPMD, ???
- Efficient parallel I/O is difficult
  - **ADIOS2**, with **openPMD** metadata

SIMD (vectorization for CPUs)

- Modern CPUs can execute up to 8 operations simultaneously
  - (“threads” in CUDA, CPU threads are “blocks” in CUDA)
- Compilers are not good at generating SIMD code from scalar kernels