A data movement-aware implementation of the a-posteriori subcell limiting ADER-DG algorithm

Dominic E. Charrier, Tobias Weinzierl

ExaHyPE is a H2020 project where an international consortium of scientists write a simulation engine for hyperbolic equation system solvers based upon the Arbitrary DERivatives Discontinuous Galerkin (ADER-DG) paradigm. Two grand challenges are tackled with this engine: long-range seismic risk assessment and the search for gravitational waves emitted by binary neutron stars. The code itself is based upon a merger of flexible spacetree data structures with highly optimised compute kernels for the majority of the simulation cells.

This talk is a tour de force through ExaHyPE. We start with a brief sketch of the project and the underlying workflow of the ADER-DG scheme. It becomes obvious that the scheme - if not implemented naively - suffers significantly from high memory and data movement demands on modern architectures. In the main part of the talk, we thus highlight various techniques how to reduce the number of data movements to and from the memory subsystem. The remainder of the talk reports on our progress with merging this data movement-aware ADER-DG implementation with Finite Volumes patch-based a-posteriori limiting of non-physical, numerical oscillations.

Durham University

School of Engineering and Computing Sciences

A data movement-aware implementation of the a-posteriori subcell limiting ADER-DG algorithm

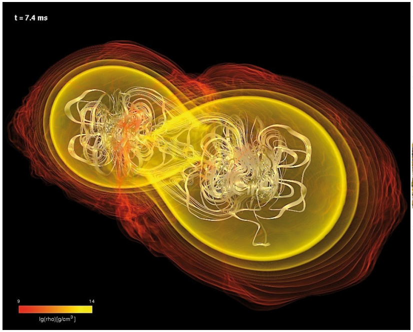
NACHOS Seminar

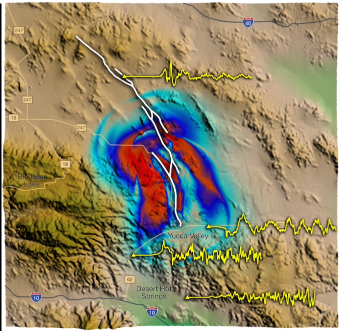
June 2017

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671698 (ExaHyPE).

Durham University
School of Engineering and Computing Sciences

An Exascale Hyperbolic PDE Engine

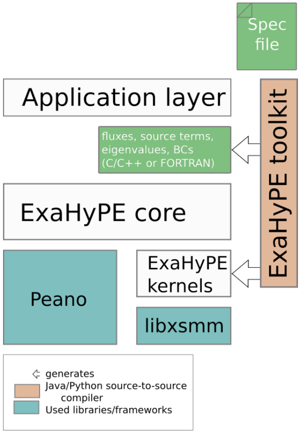




- Two grand challenges (seismic risk assessment and gravitational waves)
- A simulation engine (cmp. 3d game engine)
- Enable groups* to write an exascale code within a year

* without extreme scale expertise but with HPC affinity

Software architecture blueprint


Spec file
Application layer
fluxes, source terms, eigenvalues, BCs (C/C++ or FORTRAN)
ExaHyPE core
ExaHyPE toolkit
Peano
ExaHyPE kernels
libxsmm
generates
Java/Python source-to-source compiler
Used libraries/frameworks

ExaHyPE architecture

Design decisions:

- First-order hyperbolic PDEs
  $\Rightarrow \partial_t u + \mathbf{divF} + \mathrm{NCP}(u, \nabla u) = f$
- One grid paradigm
  $\Rightarrow$ adaptive Cartesian meshes
- One numerics paradigm
  $\Rightarrow$ ADER-DG with FV limiter

Usage:

- User code focuses on model/math
  “What” is done not “how”
- Source-to-source compiler integrates user code into architecture
- Architecture provides efficiency and parallelism
  MPI+TBB / MPI+OpenMP / MPI+TBB+CUDA
- For particular setups, we offer tailored compute kernels (for fluxes) through precompiler
  vectorisation and shared memory efficiency

An ExaHyPE spec file

```
exahype-project  Euler
  peano-kernel-path const = ./Peano
  exahype-path const      = ./ExaHyPE
  output-directory const  = ./ApplicationExamples/EulerFlow
  architecture const      = noarch
  log-file                = mylogfile.log

  computational-domain
    dimension const = 2
    width           = 1.0, 1.0
    offset          = 0.0, 0.0
    end-time        = 0.5
  end computational-domain

  solver ADER-DG MyEulerSolver
    variables const    = rho:1,j:3,E:1
    order const        = 3
    maximum-mesh-size  = 0.15
    time-stepping      = global
    kernel const       = generic::fluxes::nonlinear
    language const     = C

    plot vtk::Cartesian::vertices::ascii  ConservedQuantitiesWriter
      variables const  = 5
      time             = 0.0
      repeat           = 0.05
      output           = ./conserved
    end plot
  end solver
end exahype-project
```

An ExaHyPE implementation

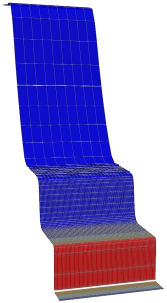
Durham University
School of Engineering and Computing Sciences

- Example: Compressible Euler equations in conservation form

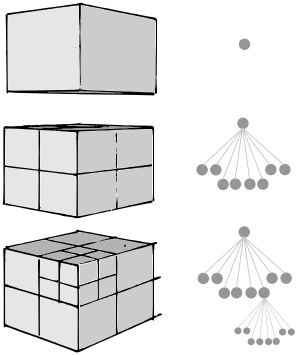
$$\partial_t \begin{pmatrix} \rho \\ \mathbf{j} \\ E \end{pmatrix} + \mathbf{div} \begin{pmatrix} \mathbf{j} \\ \frac{1}{\rho}\mathbf{j} \otimes \mathbf{j} + p \cdot \mathbf{I} \\ \frac{1}{\rho}(E+p) \cdot \mathbf{j} \end{pmatrix} = 0, \qquad p = (\gamma - 1) \cdot \left(E - 0.5\,\frac{1}{\rho}\,\mathbf{j} \cdot \mathbf{j} \right)$$

```
void Euler::MyEulerSolver::flux(const double* const Q, double** F) {
  ReadOnlyVariables vars(Q);
  Fluxes f(F);

  tarch::la::Matrix<3,3,double> I;
  I = 1, 0, 0,
      0, 1, 0,
      0, 0, 1;

  constexpr double GAMMA = 1.4;
  const double irho = 1./vars.rho();
  const double p = (GAMMA-1) * (vars.E() - 0.5 * irho * vars.j()*vars.j() );

  f.rho ( vars.j()                                       );
  f.j   ( irho * outerDot(vars.j(),vars.j()) + p*I );
  f.E   ( irho * (vars.E() + p) * vars.j()              );
}
```

Outline



Spacetree grids



- Generalisation of quadtrees/octrees to facilitate dynamic AMR
- Horizontal cut-through prescribes DG cells *and/or*
- Horizontal cut-through hosts regular Cartesian Finite Volume patch

  Tree acts as a meta/helper data structure carrying one or multiple compute meshes
- Mesh traversal based upon pushback automaton with space-filling curves

  Peano framework—`www.peano-framework.org`

ADER-DG

- We support well-known ("cumbersome") Cauchy-Kowalewsky procedure based formulation for linear PDEs
- As well as novel ADER-DG formulation by Michael Dumbser et al. which is able to treat nonlinear PDEs; see [1] for a description of both formulations
- We limit non-physical oscillations via Finite Volumes based a-posteriori limiting [2]
- Novel Formulation by Michael Dumbser et al.:
  1. Locally implicit solve via Picard iterations:

$$(\partial_t q_h, \varphi_h)_{(K\times I_K)} + (\mathbf{div}\mathbf{F}, \varphi_h)_{(K\times I_K)} = 0$$

     (q_h: space-time predictor, φ_h: space-time test function)
  2. Determine a numerical normal flux $\mathbf{G}({q_h}^+, {q_h}^-)\mathbf{n}$ by using exact or approximate Riemann solvers
  3. Riemann solve + DG update:

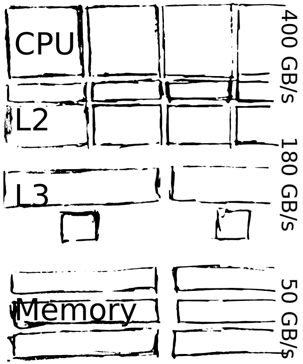
$$(\Delta u_h, v_h)_K = -\,(\mathbf{F}(q_h), \mathrm{grad} v_h)_{(K\times I_K)} + \left(\mathbf{G}({q_h}^+, {q_h}^-)\mathbf{n}, v_h\right)_{(\partial K\times I_K)}$$

     (Δu_h: solution update (spatial ansatz), v_h: spatial test function)

[1] Dumbser,M. , Balsara, D. S. , Toro, E. F. , Munz, C. -D. (2008)

[2] Dumbser, M. , Zanotti, O. , Loubre, R. , Diot, S. (2014)

Challenges

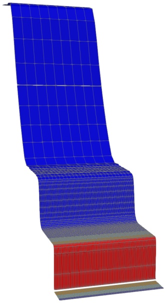


- ADER-DG challenges
  - Phases with high arithmetic intensity (STP) are followed by steps with low arithmetic intensity
  - Substeps are synchronised with each other
  - Multiple (partial) data reads (3–6) per time step
  - Massive space-time memory footprint
  - Explicit time stepping requires global synchronisation in Riemann solve (CFL conditions)
- Hardware challenges
  - Limited/decreasing memory per core and access to memory is slow

  $\Rightarrow$ Reduce data transfer to and from memory
  - Manycore chips on the rise

  $\Rightarrow$ Multicore-ise code lightweightly

Outline



ADER-DG revisited

Durham University
School of Engineering and Computing Sciences

Step 1 Locally implicit solve:

$$(\partial_t q_h, \varphi_h)_{(K\times I_K)} + (\mathbf{divF}, \varphi_h)_{(K\times I_K)} = 0$$

(q_h: space-time predictor, φ_h: space-time test function)

```
for (int ix=0; ix<nx; ix++)
for (int iy=0; iy<ny; iy++) {   // loop over cells
 STP(
  cell(ix,iy), xface(ix,iy),xface(ix+1,iy),
  yface(ix,iy),yface(ix,iy+1) );
}
```

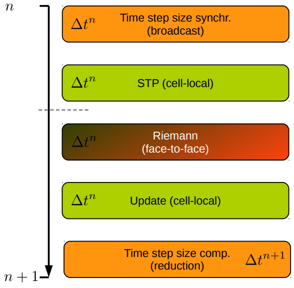
Step 2+3 Riemann solve + DG update:

$$(\Delta u_h, v_h)_K = -(\mathbf{F}(q_h), \text{grad} v_h)_{(K\times I_K)} + \left(\mathbf{G}({q_h}^+, {q_h}^-)\mathbf{n}, v_h\right)_{(\partial K\times I_K)}$$

(Δu_h: solution update (spatial ansatz), v_h: spatial test function)

```
for (int ix=0; ix<nx+1; ix++)
for (int iy=0; iy<ny; iy++) {   // loop over faces
 Riemann(xface(ix,iy),xface(ix+1,iy));
}
...
for (int ix=0; ix<nx; ix++)
for (int iy=0; iy<ny; iy++) {   // over cells again
 Update( cell(ix,iy),
  xface(ix,iy),xface(ix+1,iy),
  yface(ix,iy),yface(ix,iy+1) );
}
```

How to break ADER-DG (down into tasks)


n
Δt^n Time step size synchr. (broadcast)
Δt^n STP (cell-local)
Δt^n Riemann (face-to-face)
Δt^n Update (cell-local)
Time step size comp. (reduction) Δt^{n+1}
$n+1$

- Decompose per-cell algorithm into its 3 main constituents (tasks)
  1. STP
  2. Riemann
  3. Update

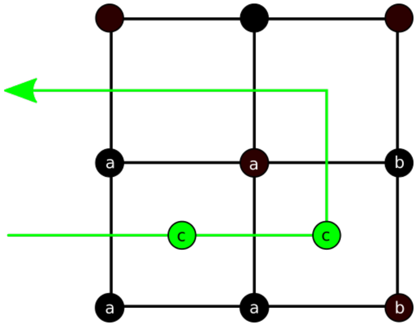
  post Limiter evaluation (perhaps)
- Observations:
  1. Noone forces us to synchronise the steps on all the grid entities

  $\Rightarrow$ break-up algorithm into steps per cell & read algorithm as task graph (DAG)
  2. It is the potential violation of the CFL condition that bothers us

  $\Rightarrow$ think positive
  3. There is no need to synchronise grid walk throughs and time steps

  $\Rightarrow$ shift operations between grid sweeps

ADER-DG decomposed

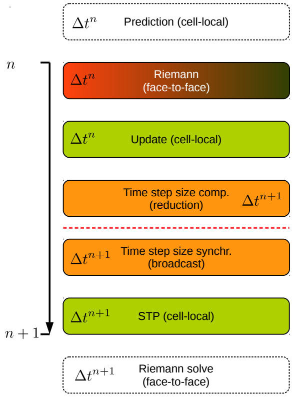


(a)+(b) Whenever we read a vertex, we lookup the 2 *d* adjacent faces and perform a Riemann solve (if not done yet)

(c) Whenever we run into a cell, all vertices have been read before

$\Rightarrow$ If we equip all faces with a marker (Riemann problem solved), we can intermix Riemann solve and Update

Tasking and pipelining

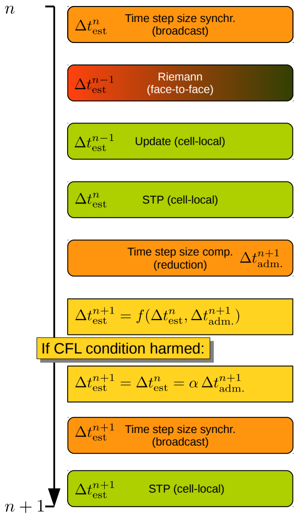


Δt^n Prediction (cell-local)
n
Δt^n Riemann (face-to-face)
Δt^n Update (cell-local)
Time step size comp. (reduction) Δt^{n+1}
Δt^{n+1} Time step size synchr. (broadcast)
Δt^{n+1} STP (cell-local)
$n+1$
Δt^{n+1} Riemann solve (face-to-face)

- Mitigate second synchronisation*:
  - Exchange of admissible $\Delta t^{(n+1)}$ mandatory
  - Work with estimate $\Delta t_{est}^{(n+1)}$
  - Start with $\Delta t_{est}^{(n+1)} = \Delta t^{(n+1)}$
  - If admissible $\Delta t_{adm}^{(n+1)} > \Delta t_{est}^{(n+1)}$ use $\Delta t_{est}^{(n+1)} \leftarrow f(\Delta t_{adm}^{(n+1)}, \Delta t_{est}^{(n+1)})$
  - If $\Delta t_{adm}^{(n+1)} < \Delta t_{est}^{(n+1)}$ reset $\Delta t_{est}^{(n+1)} \leftarrow \alpha\, \Delta t_{adm}^{(n+1)}$

$\Rightarrow$ Two passes (worst case) instead of three

Tasking and pipelining

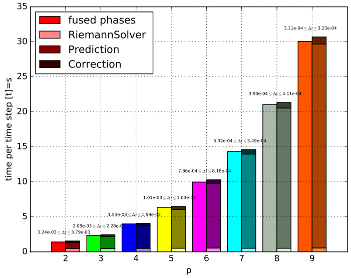

n
$\Delta t_{\text{est}}^{n}$ Time step size synchr. (broadcast)
$\Delta t_{\text{est}}^{n-1}$ Riemann (face-to-face)
$\Delta t_{\text{est}}^{n-1}$ Update (cell-local)
$\Delta t_{\text{est}}^{n}$ STP (cell-local)
Time step size comp. (reduction) $\Delta t_{\text{adm.}}^{n+1}$
$\Delta t_{\text{est}}^{n+1} = f(\Delta t_{\text{est}}^{n}, \Delta t_{\text{adm.}}^{n+1})$
If CFL condition harmed:
$\Delta t_{\text{est}}^{n+1} = \Delta t_{\text{est}}^{n} = \alpha\, \Delta t_{\text{adm.}}^{n+1}$
$\Delta t_{\text{est}}^{n+1}$ Time step size synchr. (broadcast)
$\Delta t_{\text{est}}^{n+1}$ STP (cell-local)
$n+1$

- Mitigate second synchronisation:
  - Exchange of admissible $\Delta t^{(n+1)}$ mandatory
  - Work with estimate $\Delta t_{est}^{(n+1)}$
  - Start with $\Delta t_{est}^{(n+1)} = \Delta t^{(n+1)}$
  - If admissible $\Delta t_{adm}^{(n+1)} > \Delta t_{est}^{(n+1)}$ use $\Delta t_{est}^{(n+1)} \leftarrow f(\Delta t_{adm}^{(n+1)}, \Delta t_{est}^{(n+1)})$
  - If $\Delta t_{adm}^{(n+1)} < \Delta t_{est}^{(n+1)}$ reset $\Delta t_{est}^{(n+1)} \leftarrow \alpha\, \Delta t_{adm}^{(n+1)}$

$\Rightarrow$ two passes (worst case) instead of three

- Next: Fuse with other phases (limiter), too

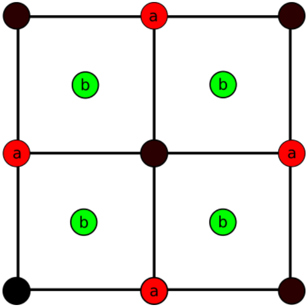
Preliminary results



$d = 2$ results for Euler flow (5 unknowns) on 2x8 core Sandy Bridge (left). 8 STP reruns during 208 time step.

- Our kernels are currently not memory bound – other than expected. Performance analysis is ongoing
- Optimisation only yields a small runtime improvement at the moment
- For small p, we remain grid admin restricted despite the optimisation
- For larger p, predictor's 'enormous' runtime dominates everything

ADER-DG on multicores



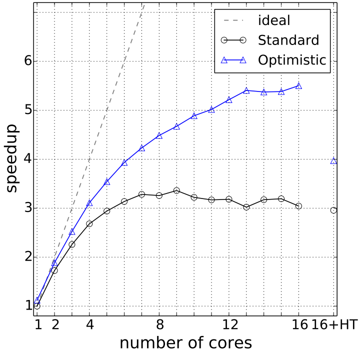
- Peano stores regular subgrids in a regular data structure – if built with shared memory support

a Riemann solves are race-free once we use **red-black** colouring

b Update plus STP are embarrassingly concurrent

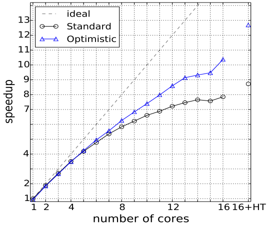
Standard vs. fused (3rd order, uniform)

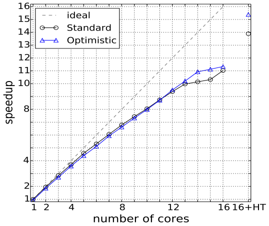
- Compressible Euler 2D on uniform grid with 27^2 cells
- Speedup of both implementations with respect to the serial runtime of the standard implementation

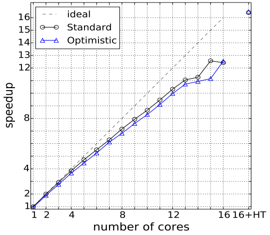


Standard vs. fused (5th,7th, and 9th order, uniform)

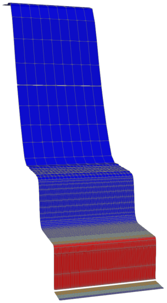
- Speedup of time to solution with respect to serial runtime of standard implementation



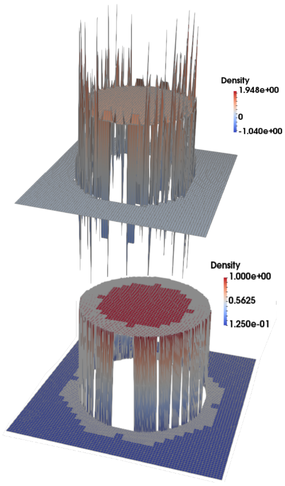




Outline



ADER-DG with a-posteriori subcell limiting (I)



Density shock: unlimited (top) vs. limited (bottom)

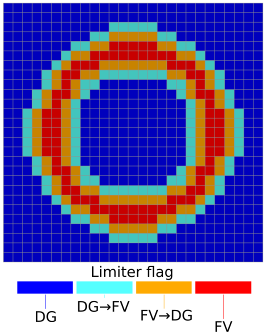
- Discontinuous initial data and shock formation can introduce spurious oscillations into the (ADER-)DG solution
- We thus recompute the solution in cells where these issues arise on a Finite Volumes (FV) subgrid
- Those troubled cells K are detected a-posteriori via a discrete maximum principle (DMP):

$$\min_{\substack{x \in K' \\ K' \in V(K)}} u_{h,i}^{n-1}(x) - \delta \leq u_{h,i}^{n}|_K(x) \leq \max_{\substack{x \in K' \\ K' \in V(K)}} u_{h,i}^{n-1}(x) + \delta$$

$i \in \{1, 2, \ldots N_{\text{var}}\}$: component of solution vector
$\delta > 0$: relaxation parameter
$V(K)$: direct neighbours of K

- Plus via user-supplied physical admissibility criteria (PAD; e.g. $\rho > 0$)

ADER-DG with a-posteriori subcell limiting (II)



- FV subgrids are of size $N = 2 \cdot p + 1$, where p is the DG order

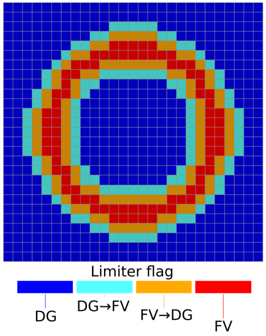
  $\Rightarrow$ CFL numbers of DG and FV are identical

- Troubled FV cells are surrounded by two helper cell layers (DG$\rightarrow$FV , FV$\rightarrow$DG) to facilitate communication between DG and FV domains
- DG$\rightarrow$FV cells compute with ADER-DG and project (valid) DG solution onto FV subgrid
- FV$\rightarrow$DG cells compute with FV and project (valid) FV solution onto DG basis
- Algorithm (non-fused):
  1. STP
  2. Riemann
  3. Update

  limit if at least one cell is troubled

  4. Communicate limiter flag
  5. FV Update in FV and FV$\rightarrow$DG cells

Modification 1: Rewrite it as hybrid solver



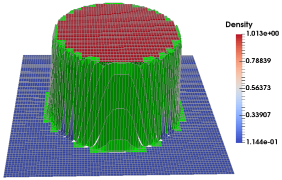
- Goal: We still want to perform all tasks within one grid traversal
- We interpret ADER-DG with a-posteriori subcell limiting as ADER-DG – FV hybrid solver
- We only go into the limit branch if solver domains change!
- FV solver is shifted by half a step similar to fused ADER-DG scheme in our implementation

  $\Rightarrow$ Riemann first, Update second
- ADER-DG - FV hybrid solver time step is realised within one grid traversal
- Algorithm:
  1. Fused ADER-DG + FV time step

  limit if ADER-DG (or FV) domain changes

  2. Communicate limiter flags (2 loops)
  3. Rollback + Rieman solve + FV update for FV and FV$\rightarrow$DG cells (3 loops)
  4. STP in FV$\rightarrow$DG cells (1 loop)

Spherical explosion on coarse grid with low order ADER-DG

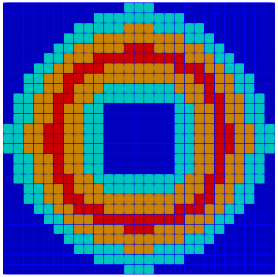


Density profile. Troubled cells are highlighted (green).

- Setting:
  - Compressible Euler equations in 2d
  - Low order $p = 3$ ADER-DG discretisation
  - Godunov FV limiter with $N = 2 \cdot p + 1$ cells per coordinate direction
  - 27×27 elements
  - $t_{\text{final}} = 0.12$
  - Worst case scenario: Spherical explosion
- The FV limiter was active in 71 of 83 time steps.
- The FV limiter domain had only to be updated 54 times

  $\Rightarrow$ 24% less limiter branch visits
- Looks better for higher orders – since CFL$\propto 1/(2 \cdot p + 1)$ – and/or finer resolutions

Modification 2: Distinguish between shock capturing and recomputation

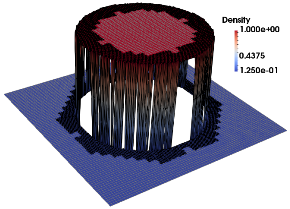

Gradual update of the limiter domain.

- Observation: If a shock wanders through the domain, the troubled cell flag moves gradually from cell to cell
- We propose to double the width of the helper cell layers (1→2) and allow gradual updates of the mesh during the (fused) time stepping iterations
  - ⇒ Gradual limiter domain changes can be accommodated by helper cells
  - ⇒ Recomputations only necessary for irregular changes
- Algorithm:
  1. Fused ADER-DG - FV time step + gradual ADER-DG & FV domain updates

  recompute if solver domains change irregularly

  2. Communicate limiter flag (4 loops)
  3. Rollback + FV Riemann solve + FV update for FV and FV→DG cells (3 loops)
  4. STP in FV→DG cells (1 loop)

Fused ADER-DG-FV hybrid solver



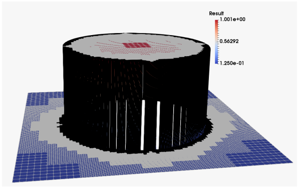
Density profile. FV and FV$\to$DG cells are highlighted by black mesh edges.

- Setting:
  - Compressible Euler equations in 2d
  - Low order $p = 3$ ADER-DG discretisation
  - Godunov FV limiter with $N = 2 \cdot p + 1$ cells per coordinate direction
  - 27×27 elements
  - $t_{\text{final}} = 0.12$
- The FV limiter was active in 71 out of 83 time steps.
- The FV limiter domain did only change irregularly 13 times

  $\Rightarrow$ 81% less limiter branch visits
- Looks better for higher orders – since CFL$\propto 1/(2 \cdot p + 1)$ – and/or finer resolutions

Combination with (dynamic) AMR

Durham University
School of Engineering and Computing Sciences



Warped density contour. Troubled (FV) cells and FV helper cells are highlighted black.

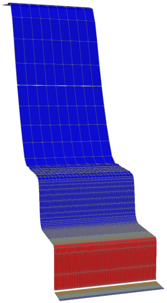
- Design decisions:
  - FV solver is only active on finest level
  - Troubled cells on coarser level, are refined down to finest level
  - Helper cell flagging on coarser levels is used to create halo layer around shock (this is no global 2:1 balancing!)

  ⇒ Implementation requires multi-scale representation of grid
- Aside from mesh refinement and local recomputation, we have additional code branch global recomputation
- If coarse grid cell becomes troubled, we then need to:
  - Perform a rollback in all cells
  - Perform mesh refinement
  - perform a complete fused ADER-DG-FV hybrid solver time step (where we do not perform limiter flagging)

Outline

Durham University

School of Engineering and Computing Sciences



Summary & outlook

Summary:

- Proposed reordering and fusing of ADER-DG algorithmic phases such that each vertex (or face) and each cell is only touched once per time step (most of the time)
- Dynamic a-posteriori limiting was integrated into the single-touch ADER-DG formulation by widening the helper cell layers

Research question:

- How much artificial diffusion do both high-level optimisations add? Is the artificial diffusion noticeable – especially if we use higher order FV limiters (MUSCL-Hancock, Central-WENO,. . .)?

Next steps:

- Performance engineering and upscaling
- Convergence tests with shock problems
- Results on completely dynamically adaptive grids running on KNLs. No statements on sustainable peak performance possible yet.
- First insights from application areas made possible by ExaHyPE.

Support & acknowledgements

- This talk picks up challenges tackled by the project ExaHyPE (www.exahype.eu). The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671698 (ExaHyPE).
- The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ, www.lrz.de).
- Experiments were made possible through Durham's supercomputer Hamilton.