Algorithmic components of ForestClaw adaptive mesh library: Multi-rate time stepping

Donna Calhoun (Boise State University)

Carsten Burstedde (Univ. of Bonn)

Collaborators: R. J. LeVeque (Univ. of Washington); M. J. Berger (NYU); D. George (USGS); D. Ketcheson (KAUST); K. Mandli (Columbia); Christiane Helzel (Univ. of Dusseldorf)

SIAM Parallel Processing
April 12-15, 2016
Universite Pierre et Marie Curie
Paris, France
Adaptive Mesh Refinement (AMR)

Overlapping patch-based AMR

Original approach (Berger, 1984)

Chombo (LBL), AMRClaw (UW, NYU), Boxlib* (LBL), SAMRAI (LLNL), AMROC (S. Hampton)

*See IP4: “Next Generation AMR (A. Almgren, Thursday 8:30)
Adaptive Mesh Refinement (AMR)

Quadtree/Octree based AMR

Quad-tree approach

p4est (U. Bonn), PARAMESH (U. Chicago), ForestClaw, Gerris (Paris VI), Racoon II (U. Bochum), RAMSES (U Zurich), Nirvana (Potsdam), "Building Cubes" (Tohoku)
Skepticism about AMR

There remains skepticism about how effective adaptive mesh refinement (AMR) can be

• Coarse/fine boundaries with abrupt resolution changes are regarded with suspicion,
• Lack of good refinement criteria dampens enthusiasm for trying out AMR,
• Not obvious how to extend sophisticated numerical algorithms and applications to the adaptive setting,

When AMR is used,
• Multi-rate time stepping is not always used,
• The goals are often modest: “Do no harm!”
• Grids are often only static; not dynamically refined.
ForestClaw Project

A parallel, adaptive library for logically Cartesian, mapped, multi-block domains

Features of ForestClaw include:

- Uses the highly scalable p4est dynamic grid management library (C. Burstedde, Univ. of Bonn, Germany)
- Each leaf of the quadtree contains a fixed, uniform grid,
- Optional multi-rate time stepping strategy,
- Has mapped, multi-block capabilities, (cubed-sphere, for example) to allow for flexibility in physical domains,
- Modular design gives user flexibility in including several solvers and packages.
- Uses essentially the same algorithmic components as patch-based AMR

ForestClaw development supported by the National Science Foundation

www.forestclaw.org
Mapped multi-block domains

- Torus
- Disk
- Annulus

Each block is a quadtree

“Brick domain”
Mapped multi-block domains

Spherical coordinates

“Pillow grid” (with C. Helzel, R. LeVeque, SIAM Review, 2009)

Cubed sphere

Each block is a quadtree
Tracer flow in a prescribed velocity field
Filling ghost cells

Each grid (or “leaf”, in p4est terminology) has one of more layers of ghost cells used for communication between grids.

Ghost cells are filled by copying from same-size neighbors, averaging from fine grid neighbors or interpolation to fine grid neighbors.

Donna Calhoun  (Boise State)
How are ghost cells filled?

- Copy
- Average
- Interpolate
Multirate time stepping

\[ t + (\Delta t)_c \]

\[ t + (\Delta t)_f \]

\[ t \]

Level 4  Level 5  Level 6  Level 7
Multi-rate time stepping

Levels: L = 3, L = 4, L = 5, L = 6

Coarsest level
Finest level

Time: t

Δt
Parallel multirate time stepping

Levels: $L=3$, $L=4$, $L=5$, $L=6$

Coarsest level

Finest level

Time

$t + \Delta t$
Multirate algorithm

Require: Grids at all levels at time $t$ must have valid ghost cells values.

for $k = 1$ to $2^{\ell_{\text{max}} - \ell_{\text{min}}}$ do
    ADVANCE_SOLUTION($\ell_{\text{max}}, (\Delta t)\ell_{\text{max}}$)
    if multirate then
        Multirate
        if $k < 2^{\ell_{\text{max}} - \ell_{\text{min}}}$ then
            Find largest integer $p \geq 0$ such that $2^p$ divides $k$.
            $\ell_{\text{time}} = \ell_{\text{max}} - p - 1$
            UPDATE_GHOST($\ell_{\text{time}} + 1$)
        end if
    else
        Global time stepping
        UPDATE_GHOST($\ell_{\text{min}}$)
    end if
end for
UPDATE_GHOST($\ell_{\text{min}}$).

procedure ADVANCE_SOLUTION(level = $\ell$, dt_stable = $\Delta t$)
    for all grids $g$ on level $\ell$ do
        Update solution $Q^{n+1} = Q^n + \Delta t F(Q^n, t_n)$.
    end for
    if $\ell > \ell_{\text{min}}$ then
        if multirate then
            if levels $\ell$ and $\ell - 1$ are time synchronized then
                ADVANCE_SOLUTION($\ell - 1, 2\Delta t$)
                TIME_INTERPOLATE($\ell - 1, t + 2\Delta t$)
            end if
        else
            ADVANCE_SOLUTION($\ell - 1, \Delta t$)
        end if
    end if
end procedure
Does this buy us anything?

- Fewer ghost cell exchanges, since not all levels exchange data at each time step,
- Fewer parallel communications
- Fewer grids to advance.
- Improved accuracy if spatial error depends on time step size

Reduced wall clock time!

Improved accuracy!
Is multirate time stepping worth it?

Assuming an equal number of grids at each level:

\[(1 + 2 + 4 + 8 + \ldots + 2^{\ell_{\text{max}}-\ell_{\text{min}}}) = (2^{\ell_{\text{max}}-\ell_{\text{min}}+1} - 1)\]

versus

\[2^{\ell_{\text{max}}-\ell_{\text{min}}} (\ell_{\text{max}} - \ell_{\text{min}} + 1)\]

on a uniformly refined grid, for a speed-up of

\[\frac{\text{multi rate}}{\text{global time step}} = \frac{2}{\ell_{\text{max}} - \ell_{\text{max}} + 1} - \frac{1}{2^{\ell_{\text{max}}-\ell_{\text{min}}} (\ell_{\text{max}} - \ell_{\text{min}} + 1)}\]

\[\approx \frac{2}{\ell_{\text{max}} - \ell_{\text{min}} + 1}, \quad \text{for large number of levels}\]

A multi-rate strategy also requires fewer parallel exchanges
Parallel Performance

- Scalar advection in a constant, periodic flow
- Uniform, replicated adaptive, and adaptive scenarios
- Fixed, global time step (160 steps total)
- Regrid every coarse grid time step (20 regrid steps)
- Four levels of refinement (levels 4-7)
- MPI processes only; no threading or tuning, and no attention paid to processor topology
- JUQUEEN BlueGene/Q (Juelich Supercomputing Center, Germany)
Cost of AMR for this problem

AMR tasks/Solving problem as a function of granularity
Replicated problem

64 processes (four per brick)

Fixed resolution
Walltime - 32x32 (replicated)

Non-multirate steps : 202,396
Multirate steps        : 172,427
Difference :             : 15%

Actual speed-up : 13%
Single adaptive problem

Increase the resolution

Increase the proc count

Donna Calhoun  (Boise State)

SIAM PP - Paris - April 15, 2016
Walltime - 32x32 (single)

Wall clock time savings (%) (Advance steps savings %)

(Smaller % is better)
Is multi-rate worth it?

- Absolutely yes. It doesn’t have to be difficult to include, and will generally be faster.
- Essential for many problems
- Savings generally depend on how much of the domain is covered in fine grids
- If spatial accuracy depends on the time step size, then multi-rate schemes should be more accurate

Caveat

- Mixed equations (hyperbolic, parabolic, elliptic) present challenge for multi-rate schemes
- Additional work may be needed to maintain conservation, accuracy at coarse/fine boundaries.
Subdivide the domain

6997.1s (1.9hr)

2 × 2 array of 512 × 512 grids

This will improve cache performance enormously
Subdivide the domain

4131.7s (69min)

4 × 4 array of 256 × 256 grids

This will improve cache performance enormously
Subdivide the domain

$3813.7\text{s (64min)}$

$8 \times 8 \text{ array of } 128 \times 128 \text{ grids}$

This will improve cache performance enormously
Subdivide the domain

16 × 16 array of 64 × 64 grids

This will improve cache performance enormously

3865.3s (64min)
Timing vs block size

9194.5s (1) (2.5 hrs)
3813.7s (2.4) (1.1 hrs)

\[
\frac{\text{fastest}}{\text{slowest}} \approx 2.5
\]
Distribute patches to hardware threads. On a four core machine, this should lead to about a factor four speedup.
Speed up using 4 cores (Intel i7)

9194.5s 9194.5s 9194.5s
3813.7s 3813.7s 3813.7s
749.4s 749.4s 749.4s

Time (s)

Block size

32 64 128 256 512 1024

(2.4) (5.0) (12.3)
Can we improve on this?

Hardware solutions: Use more computing units (cpus/threads/cores)
-- Eventually, however, the communications costs will overwhelm any gains made in subdividing the domain further
-- Problems always outgrow the hardware that is available

Software solutions: Reduce the “factor 8” scaling law and put resources only where they are needed.
-- This is much more difficult than just adding more computing units
-- Should complement hardware advances
Effective 512 x 512 resolution (116s vs. 917s)
With adaptive mesh refinement

Almost a factor of 15 improvement!
One more tweak ...

In two dimensions, if we double the resolution, we increase the computational cost by a factor of 8

$$\text{(Number of grid cells increases by a factor of 4)} \times \text{(Number of time steps increase by a factor 2)} = \text{(8-fold increase in computational expense)}$$

The CFL condition constrains the size time step we can take on a grid. For stability, smaller grid cells need smaller time steps.
Multi-rate time stepping

Almost a factor of 30 improvement!
Scaling of single adaptive problem

Mimic scaling of uniform problem: Increase resolution and processor count simultaneously.
### Adaptive scaling

#### Strong scaling
- Keep minimum level fixed;
- Increase maximum refinement level

<table>
<thead>
<tr>
<th>mx = 8</th>
<th>4-7</th>
<th>4-8</th>
<th>4-9</th>
<th>4-10</th>
<th>4-11</th>
<th>4-12</th>
<th>4-13</th>
<th>4-14</th>
<th>4-15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5058</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4</td>
<td>1264</td>
<td>2876</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>16</td>
<td>316</td>
<td>719</td>
<td>1579</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>64</td>
<td>79</td>
<td>179</td>
<td>394</td>
<td>764</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>256</td>
<td>19</td>
<td>44</td>
<td>98</td>
<td>191</td>
<td>239</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1024</td>
<td>---</td>
<td>11</td>
<td>24</td>
<td>47</td>
<td>59</td>
<td>75</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4096</td>
<td>---</td>
<td>---</td>
<td>6</td>
<td>11</td>
<td>14</td>
<td>18</td>
<td>24</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>16384</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>65536</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

#### Weak scaling
- Fixed number of coarse grid time steps; final time depends on grid size.
- Strong scaling results similar to those for replicated problem.
Strong scaling - adaptive - 8x8

Time in advance

Torus (Juqueen): Advance (8x8; adaptive)

Total walltime

Torus (Juqueen): Walltime (8x8; adaptive)
Weak scaling - adaptive

Adaptive (non-replicated) results

Uniform results
## Weak scaling results

### Uniform/replicated scaling

<table>
<thead>
<tr>
<th>mx = 8</th>
<th>4-7</th>
<th>4-8</th>
<th>4-9</th>
<th>4-10</th>
<th>4-11</th>
<th>4-12</th>
<th>4-13</th>
<th>4-14</th>
<th>4-15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5058</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4</td>
<td>1264</td>
<td>2876</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>16</td>
<td>316</td>
<td>719</td>
<td>1579</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>64</td>
<td>79</td>
<td>179</td>
<td>394</td>
<td>764</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>256</td>
<td>19</td>
<td>44</td>
<td>98</td>
<td>191</td>
<td>239</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1024</td>
<td>---</td>
<td>11</td>
<td>24</td>
<td>47</td>
<td>59</td>
<td>75</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4096</td>
<td>---</td>
<td>---</td>
<td>6</td>
<td>11</td>
<td>14</td>
<td>18</td>
<td>24</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>16384</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>---</td>
</tr>
<tr>
<td>65536</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>---</td>
</tr>
</tbody>
</table>

### Grids per processor, as percent of grids on processor on original problem.

---

Donna Calhoun  (Boise State)
Adaptive scaling - 8x8

Plot timing results as function of granularity
Adaptive scaling - 16x16

Plot timing results as function of granularity
Adaptive scaling - 32x32

Fraction time in advance/AMR (32x32; adaptive)

Plot timing results as function of granularity
Is AMR really worth it?
Skepticism about AMR

Still, there remains skepticism about how effective adaptive mesh refinement (AMR) can be

• Coarse/fine boundaries with abrupt resolution changes are regarded with suspicion,
• Lack of good refinement criteria dampens enthusiasm for trying out AMR,
• Not obvious how to extend sophisticated numerical algorithms and applications to the adaptive setting,

When AMR is used,
• Multi-rate time stepping is not always used,
• The goals are often modest :“Do no harm!”
• Grids are often only static; not dynamically refined.
Why are AMR codes difficult to write and use?

- Heterogeneous data structures associated with storing hierarchy of grids,
- Dynamically creating and destroying grids:
  - Need a “factory” paradigm to create new grids and any auxiliary data arrays (material properties, metric terms, bathymetry, etc) that go with each new grid,
- Parallel load balancing
- Coupling of multiple solvers (e.g. advection + diffusion + elliptic solvers + source terms),
- Multi-physics,
- Data visualization and post-processing
- Computing diagnostics on a nested grid hierarchy,
- Error estimation, tuning for efficient use of grids, ...
What makes AMR codes difficult to use?

- Time stepping beyond just single step explicit schemes (IMEX, SSP, RK, ...)
- Understanding how overall time stepping interacts with dynamic grid creation, destruction and management.
- Implicit solvers,
- Coupling of multiple solvers (e.g. advection + diffusion + elliptic solvers + source terms),
- Multi-physics,
- Data visualization and post-processing
- Computing diagnostics on a nested grid hierarchy,
- Error estimation, tuning for efficient use of grids, ...
Scientific Software Development
Thanks!

25 years ago, code development began

Today

Hardhat required!

*The Big Bang Theory

Scientific Software Development …
Existing AMR Codes
What if you have ideas about

• Multi-stage, multi-step, IMEX, SSP, parallel-in-time, exponential integrators, and other time stepping schemes in an adaptive setting,
• Accuracy of multi-rate schemes for PDEs with mixed elliptic/parabolic/hyperbolic terms.
• Elliptic and parabolic solvers (iterative? direct? Explicit? Fast multipole?)
• Parallelism in the AMR setting?
• Error estimation
• Higher order accuracy
• Complex physics

Should you write yet-another-AMR code?
What they might be thinking...

We’ll just refine the grid everywhere

My grant doesn’t pay for software development

I know somebody who tried it (in 1992) and it didn’t work

It year(s) of effort it will take won’t pay off

Or...

They’ve actually tried it...
Block-structured AMR codes

- General purpose (freely available) block-structured codes
  - PARAMESH (NASA/Goddard)
  - SAMRAI (Lawrence Livermore National Lab)
  - BoxLib (Lawrence Berkeley Lab)
  - Chombo (Lawrence Berkeley Lab)
  - AMRClaw (University of Washington/NYU)
- All are large frameworks, with many developers
- Mostly C++ and Fortran libraries (no GUIs) that started life as research codes.

See my website for a list of many more application specific codes
AMR Software

“PARAMESH is a package of Fortran 90 subroutines designed to provide an application developer with an easy route to extend an existing serial code which uses logically Cartesian structured mesh into a parallel code with adaptive mesh refinement”

SAMRAI - “Object oriented C++ library developed to provide algorithmic and software support to large scale multiphysics problems relevant to the US Department of Energy (DOE)”

Boxlib - “These libraries provide the software infrastructure for the computational activities (combustion, astrophysics, porous media flow) at the Center for Computational Sciences and Engineering (CCSE) at Lawrence Berkeley Labs”
Mental model of how this might work:

* Idea for code name: OmNum
The Dream

AMR.run(max_time, max_steps);

Your single grid solver is called from here.
Block-structured AMR

The Reality

```cpp
Tuple< RefCountedPtr<AMRLevelOpFactory<
    LevelData<FArrayBox> >>>, SpaceDim> velTGAOpFactoryPtrs;

for (int idir = 0; idir < SpaceDim; idir++)
    {velTGAOpFactoryPtrs[idir] =
        RefCountedPtr<AMRLevelOpFactory<LevelData
            <FArrayBox> >>>
        ((AMRLevelOpFactory<LevelData<FArrayBox> >*)
            (new AMRPoissonOpFactory())); //.....
```

Your single grid solver

you
node(ndjhi,mptrnx) = node(ndjhi,mptr)
node(ndjhi,mptr) = node(ndjlo,mptr) + nyl - 1
node(ndjlo,mptrnx) = node(ndjhi,mptr) + 1
node(ndihi,mptrnx) = node(ndihi,mptr)
node(ndilo,mptrnx) = node(ndilo,mptr)

rnode(cornxlo,mptrnx) = cxlo
rnode(cornylo,mptrnx) = cymid
rnode(cornyhi,mptrnx) = cyhi
rnode(cornxhi,mptrnx) = cxhi
node(nestlevel,mptrnx) = node(nestlevel,mptr)
rnode(timemult,mptrnx) = rnode(timemult,mptr)
go to 10
Most of these codes were designed mainly to support large scale applications developers and in-house scientific research.

Also initially designed largely for hyperbolic PDEs and single step methods.
Natural Hazards Modeling
Finite volume discretization
Modeling physical processes - Finite volumes

A model of the transport of $q(x, t)$ in a control volume $C$

$$\frac{d}{dt} \int_C q(x, t) \, dA = - \int_C \nabla \cdot f(q(x, t)) \, dA$$

$$= - \int_{\partial C} f(q(x, t)) \cdot n \, dL$$
Physical fluxes

Fluxes depend on underlying physical model and quantities being conserved, i.e.

- mass, tracer species (ash, smoke, …) \( f(q) = uq \)
- heat \( f(q) = -\beta q_x \)
- momentum \( f(\rho, u, E) = u^2 \rho + P(\rho, u, E) \)
- energy \( f(\rho, u, E) = u(E + P(\rho, u, E)) \)

By updating quantities using fluxes, we can guarantee numerical conservation of mass, energy, momentum, tracers, etc.
Using numerical fluxes, we use the explicit update formula:

\[
Q_{i}^{n+1} = Q_{i}^{n} - \frac{\Delta t}{\Delta x} \left[ F_{i+1/2}^{n} - F_{i-1/2}^{n} \right]
\]

where

\[
F_{i-1/2}^{n} \approx \frac{1}{\Delta t} \int_{t_{n}}^{t_{n+1}} f(q(x_{i-1/2}, t)) \, dt
\]
We assume an explicit time stepping scheme

\[ F^n_{i-1/2} \approx \int_{t_n}^{t_{n+1}} f(q(x_{i-1/2}, t)) \, dt \]

so that we try to find formulas for the flux of the form

\[ F^n_{i-1/2} = \mathcal{F}(Q^n_i, Q^n_{i-1}) \]
Riemann Problems

At each cell interface, solve the hyperbolic problem with special initial data, i.e.

\[ q_t + f(q)_x = 0 \]

subject to

\[ q(x, 0) = \begin{cases} Q_{i-1} & x < x_{i-1/2} \\ Q_i & x > x_{i-1/2} \end{cases} \]
Finite volume update

\[ t = 0 \]

\[ x_{i-1/2} \]

\[ Q_{i-1} \]

\[ Q_i \]

\[ F_{i-1/2}^n = f(q^*) \]

\[ t > 0 \]

\[ x_{i-1/2} \]

\[ Q_{i-1} \]

\[ Q_i \]
Godunov schemes

Numerical flux at cell interface is then approximated by

\[ F_{i-1/2} = f(q^*) \]

This is the classical Godunov approach for solving hyperbolic conservation laws.

- Resolves shocks and rarefactions and is conservative
In one time step, conserved quantities can be transported across corners, so corner coupling must be taken into account.
Approximate Riemann solvers
Conditions across shocks

Across shocks, the left and right states must satisfy the Rankine-Hugoniot conditions

\[ f(q_r) - f(q_\ell) = s(q_r - q_\ell) \]

This leads to a nonlinear system for intermediate states.
Approximate Riemann solvers

\[ q_t + f(q)_x = 0 \]

We solve a linearized system at each cell interface, at each time step

\[ q_t + f'(\hat{q})q_x = 0 \quad \rightarrow \quad q_t + Aq_x = 0 \]

for “Roe averaged” values \( \hat{q} \).

We use the eigenvalue/eigenvector decomposition of the flux Jacobian

\[ A = R\Lambda R^{-1} \]

\[ R = [r^1, r^2, \ldots r^m] \quad \Lambda = \text{diag}(\lambda^1, \lambda^2, \ldots \lambda^m) \]
Eigenvalue decomposition

\[ q_t + A q_x = 0 \quad \rightarrow \quad q_t + R\Lambda R^{-1} q_x = 0 \]

Define characteristic variables \( \omega \in R^m \) as

\[ \omega(x, t) = R^{-1} q(x, t), \quad \omega(x, 0) = R^{-1} q(x, 0) \]

Characteristic equations decouple into \( m \) scalar equations:

\[ \omega_t^p + \lambda^p \omega_x^p = 0, \quad p = 1, 2, \ldots, m \]

Solution for \( q(x,t) \) can be obtained from the solution to characteristic equations are given by

\[ \omega^p(x, t) = \omega^p(x - \lambda^p t, 0) \]
Solution to linearized problem

For piecewise constant initial data, evolution looks like:

Decompose jump in $Q$ at the interface into waves:

$$q^* = Q_{i-1} + \alpha^1 r^1 = Q_i - \alpha^3 r^3 - \alpha^2 r^2$$

where $\alpha = R^{-1}(Q_i - Q_{i-1})$
Roe solvers

• Scheme is conservative if linearization is chosen correctly
• Much more efficient than solving the Riemann problem exactly (assuming analytic eigenvalue information is known).
• Gives accurate results, although may require a fix for transonic rarefaction waves
• Use “wave limiters” to suppress spurious oscillations

Wave propagation algorithms implemented in Clawpack (R. J. LeVeque) using “fluctuations”. This approach handles both conservative and nonconservative terms in a uniform fashion.

www.clawpack.org
Multirate time stepping
Mappings and multiblock transforms
Assume each quadrilateral mesh cell can be approximated by a ruled surface.

\[ S(\xi, \eta) = a_{00} + a_{01}\xi + a_{10}\eta + a_{11}\xi\eta, \quad 0 \leq \xi, \eta \leq 1 \]

where \( a_{ij} \in \mathbb{R}^{2\times1} \) are computed from mesh cell vertices.

- Covariant basis vectors: \( \tau(1) \approx S_\xi \) and \( \tau(2) \approx S_\eta \)
- Mesh cell areas: \( \| S_\xi \times S_\eta \| \)
- Edge normals and tangents
- Mappings are supplied as functions

\[ [xp, yp, zp] = \text{mapc2m}(xc, yc) \]
Mapping contexts

```c
fclaw2d_map_context_t* fclaw2d_map_new_squareddisk(const double alpha) {
    fclaw2d_map_context_t *cont;
    cont = FCLAW_ALLOC_ZERO (fclaw2d_map_context_t, 1);
    cont->query = fclaw2d_map_query_squareddisk;
    cont->mapc2m = fclaw2d_map_c2m_squareddisk;
    cont->user_double[0] = alpha;
}
```

No need to rely on a fixed argument list; arguments specific to the mapping can be passed into the mapping routine.

```c
call mapc2m(xc,yc,xd1,yd1,zd1)
call fclaw2d_map_c2m(map_context,blockno,xc,yc,xd1,yd1,zd1)
```

```c
static void fclaw2d_map_c2m_squareddisk(fclaw2d_map_context_t *cont, int blockno,
                                            double xc, double yc,
                                            double *xp, double *yp, double *zp)
{
    double alpha = cont->user_double[0];
    mapc2m_squareddisk(&blockno,&xc,&yc,xp,yp,zp,&alpha);
}
```
Index transformations

c    # Exchange at left neighbor
if (idir .eq. 0) then
  do  j = 1,my
    do  ibc = 1,mbc
      do  mq = 1,meqn
        if (iface .eq. 0) then
          # Left face
          i1 = 1-ibc
          j1 = j
        elseif (iface .eq. 1) then
          # Right face
          i1 = mx+ibc
          j1 = j
        endif
        call fclaw2d_transform_face(i1,j1,i2,j2,
                      &
                      qthis(mq,i1,j1) = qneighbor(mq,i2,j2)
        enddo
      enddo
    enddo
  enddo
enddo

This is the code used for every exchange at a left edge even if it is not a block boundary.

Transform_face maps (i1,j1) ghost cell values to neighbor values, using information about block boundary orientations encoded in transform_ptr.
Misc