ForestClaw: Mapped, multiblock adaptive quadtrees

Donna Calhoun (Boise State University)

Carsten Burstedde (Univ. of Bonn)

HPC³
November 9-13, 2014
KAUST - Saudi Arabia
The ForestClaw Project

- Project to develop highly scalable adaptive mesh refinement (AMR) code that combines quadtree data structures with patch based refinement
- Uses the p4est (Carsten Burstedde, Univ. of Bonn) dynamic grid management library
- Explicit solver algorithms are those described by Berger-Oliger in 1984. Implicit solvers to be added.

Design goal: Develop an easily accessible, highly scalable AMR framework to spur development of novel numerical methods and applications for AMR.
AMR using ForestClaw

q(2) at time 1.0000
AMR using ForestClaw

Quadtree based refinement

- Fine level grids
- Coarse level grids

Each grid has a layer of ghost cells (not shown) to facilitate communication
Progress on ForestClaw

ForestClaw started at [HPC]$^3$ 2012.

Progress to date
• Multi-rate time stepping (Fall, 2012)
• General mappings available (Fall, 2012)
• MPI parallel capabilities implemented (Summer, 2013)
• Build system using GNU Autotools (Summer, 2014)
• Mapped, multiblock capabilities for several useful domains, including cubed disk and sphere (Summer, 2014)
• Periodic domains (Fall 2014)
Gridding strategies

Compare patch-based vs. quad tree on this solution
Filament example

- Solve $q_t + \mathbf{u} \nabla q = 0$, $\mathbf{u} = \nabla \times \psi$, $\psi(x, y) = (4/3)r^3$.
- *Domain*: $[0, 2] \times [0, 2]$
- Fixed $\Delta t = 4 \times 10^{-3}$ for $\Delta x = 1/16$ (CFL approx. 0.74).
- Run to $T = 8.0$
- Refine to an effective resolution of $512 \times 512$ and $1024 \times 1024$.
- Use multirate time stepping
- Regrid at least every coarse grid time step
- Use MC limiter
Filament example (ForestClaw)

ForestClaw: $t = 8.00$

512x512 grid (mx=64)
Filament example (AMRClaw)

AMRClaw : t = 8.0

mx = 512 (max Id=80)
Gridding strategies (uniform)

Levels 0 (max1d=80) 1751s

Levels 3 (mx=64) 1671s
Gridding strategies (512)

Levels 0-3 \((\text{max1d}=80)\)

617s

Levels 2-3 \((\text{mx}=64)\)

1178s

256x256 effective resolution
Gridding strategies (512)

Levels 0-3 (max1d=80)
617s

Levels 3-4 (mx=32)
904s

256x256 effective resolution
Gridding strategies (512)

Levels 0-3 (max 1d=80)

617s

Levels 3-5 (mx=16)

777s

128x128 effective resolution
Gridding strategies (512)

Levels 0-3 (max1d=80)
617s

Levels 3-6 (mx=8)
811s
Gridding strategies (512)

Levels 0-3 (max 1d=40)
720s

Levels 3-6 (mx=8)
811s
Gridding strategies (512)

Levels 0-2 (maxId=40; factor 4)
636s

Levels 3-6 (mx=8)
811s
Gridding strategies (1024)

Levels 0-2 (R Factor : 4,4)

3212s

Levels 3-7 (mx=8)

3567s

64x64 effective resolution
Gridding strategies

Levels 0-3 (R Factor : 4,4,2)  
3891s

Levels 3-7 (mx=8)  
3567s
Gridding strategies

Levels 0-5 (R Factor : 2)

4060s

Levels 3-7 (mx=8)

3567s

64x64 effective resolution
<table>
<thead>
<tr>
<th>Example</th>
<th>AMRClaw</th>
<th>ForestClaw</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1751</td>
<td>1671</td>
</tr>
<tr>
<td>1</td>
<td>617</td>
<td>1178</td>
</tr>
<tr>
<td>2</td>
<td>904</td>
<td>96.8%</td>
</tr>
<tr>
<td>3</td>
<td>777</td>
<td>83.0%</td>
</tr>
<tr>
<td>4</td>
<td>811</td>
<td>70.7%</td>
</tr>
<tr>
<td>5</td>
<td>720</td>
<td>21.9%</td>
</tr>
<tr>
<td>6</td>
<td>636</td>
<td>7.5%</td>
</tr>
<tr>
<td>7</td>
<td>3212</td>
<td>8.4%</td>
</tr>
<tr>
<td>8</td>
<td>3891</td>
<td>24.9%</td>
</tr>
<tr>
<td>9</td>
<td>4060</td>
<td>27.1%</td>
</tr>
</tbody>
</table>

512x512 grid: Uniform refinement (ForestClaw 5% faster)

512x512 effective resolution (AMRClaw about 12% faster)

1024x1024 effective resolution (ForestClaw and AMRClaw are within about 15% of each other)

AMRClaw prefers larger refinement ratio; ForestClaw prefers smaller grids

Timings all done on a desktop
Why use quadtrees?

- Refinement patches and parallel “units” are the same.
- Tree-based grid layout makes communication between patches and blocks much simpler, especially in 3d.
- Has the aesthetic appeal of a quad/octree refinement.
- Make use of existing literature on make quad/octrees efficient and scalable.
What next?

- Grid exchanges and communication
- Grid mappings and multiblock
- Multi-rate time stepping
Multiblock capabilities are "inherited" from p4est.
Sphere domains
Periodic brick domains
Multiblock disk
Preservation of functional relationship between tracers.

## Mixing diagnostics (256 x 256)

- **Real mixing (r)**: 
  - Diagnostic: $5.45 \times 10^{-4}$
  - Fraction: 0.7565

- **Range preserving mixing (g)**: 
  - Diagnostic: $1.49 \times 10^{-4}$
  - Fraction: 0.2070

- **Under and over shoots (b)**: 
  - Diagnostic: $2.63 \times 10^{-5}$
  - Fraction: 0.0365

![Graph showing mixing diagnostics with and without AMR](image-url)
Flow on a torus

Cool! Periodic boundary conditions!
Handling multiblock boundaries

```fortran
if (idir .eq. 0) then
  do j = 1,my
    do ibc = 1,mbc
      do mq = 1,meqn
        if (iface .eq. 0) then
          i1 = 1-ibc
          j1 = j
        elseif (iface .eq. 1) then
          i1 = nx+ibc
          j1 = j
        endif
        call fclaw2d_transform_face(i1,j1,i2,j2,
                                  transform_ptr)
        qthis(mq,i1,j1) = qneighbor(mq,i2,j2)
      enddo
    enddo
  enddo
enddo
```

This is the code used for every exchange at a left edge even if it is not a block boundary. The `transform_face` function maps `(i1,j1)` ghost cell values to neighbor values, using information about block boundary orientations encoded in `transform_ptr`. 

Loop over all ghost cells at left edge
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
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);
}
```
Current practical limitations

Multiblock limitations
• Users cannot yet design their own block connectivity
  - p4est needs to first set up the connectivity
  - Need a file format for specifying general connectivities
• At most four blocks can meet at a corner

Solver limitations
• Metric terms limit solutions to second order (most likely)
• Solution errors at mapping seams not properly handled.

Software
• Visualization done in Matlab (no VisClaw, Visit, etc support yet)
• Regridding done only at coarse time step, not at intermediate time steps.
Future?

- Extend code to 3d
- Parallel performance testing
- Better treatment of metric terms
- Documentation

Recent NSF funding to
- Develop multistage methods in an AMR Framework (D. Ketcheson)
- Extend existing Ash3d code to AMR (USGS, D. George)
Alpha testers needed!

Let me know if you’d like to out ForestClaw
- Test installation procedure
- Run existing examples
- Port Clawpack examples
- Get some VisClaw graphics support?

No prior experience needed!
Thanks to David Ketcheson, Heba, Ting and KAUST for organizing hosting this event!