Titanium Chombo

vs

C++/Fortran Chombo

Part I

Tong Wen

Phil Colella

LBNL ANAG Group
EECS Titanium Group

March 11, 2004

Overview

1. Adaptive Mesh Re£nement (AMR), C++/Fortran Chombo and Titanium Chombo.

2. The testing problem.

3. Pro£ling results: Titanium Chombo vs. C++/Fortran Chombo.

4. Performance improvement and portability.

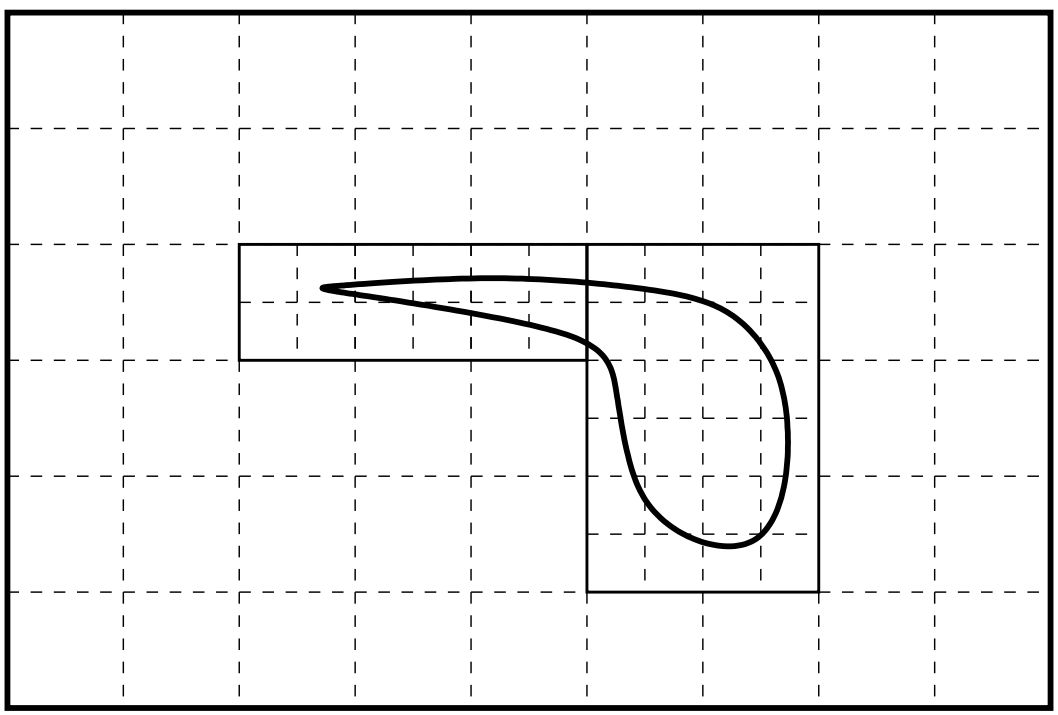
5. Future work.

Adaptive Mesh Re£nement

- Motivation:

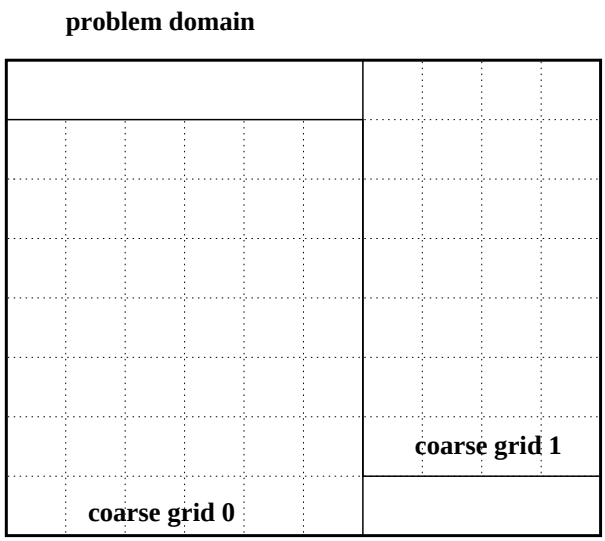
$$\begin{cases} Lu &= f \quad \text{on } U \\ u &= g \quad \text{in } \partial U. \end{cases}$$

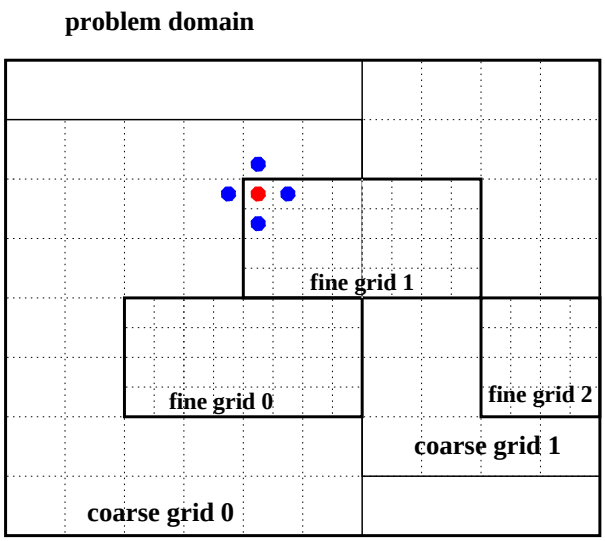
problem domain



Adaptive Mesh Re£nement

- The trade-off:
  1. Mathematically.
  2. Computationally.





Titanium Chombo

---

1. ■ BoxTools.
   - BoxLayout, **BoxedArray**, **BoxLayoutData**;
   - LayoutIndex, DataIndex⋆, LayoutIterator, DataIterator;
   - **Copier**, Util, LayoutReader.
2. ■ AMRTools.
   - **CoarseAverage**, **ConstantFineInterp**;
   - **LevelFluxRegister**, **QuadCFInterp**.
3. ■ AMRElliptic.
   - AMRSolver;
   - LevelOp, PoissonOp⋆.
4. □ AMRTimeDependent.

Titanium Chombo

```
final local void mgRelax(int single a_level){
      int single l=a_level;
      int single i,j;

      for (i=0;i<m_numSmoothDown;i++)
       smooth(mg_phi[l],mg_rhs[l],mg_dx[l],mg_CFinterp[l],l);

      if (l>1){
         mg_phi[l-1]<<=0;
         mg_phi[l-1].exchange();applyHPhyBC(mg_phi[l-1],l-1);
         mg_CFinterp[l].coarseFineInterp(mg_phi[l]);

         residual(mg_phi[l],mg_rhs[l],mg_dx[l],mg_R[l]);
         mg_Average[l].averageToCoarse(mg_rhs[l-1], mg_R[l]);

         mgRelax(l-1);

         mg_Ipwc[l].interpToFine(mg_R[l],mg_phi[l-1]);
         mg_phi[l]+=mg_R[l];
         mg_phi[l].exchange();applyHPhyBC(mg_phi[l],l);

         for (i=0;i<m_numSmoothUp;i++)
          smooth(mg_phi[l],mg_rhs[l],mg_dx[l],mg_CFinterp[l],l);
      }
}
```

The testing problem

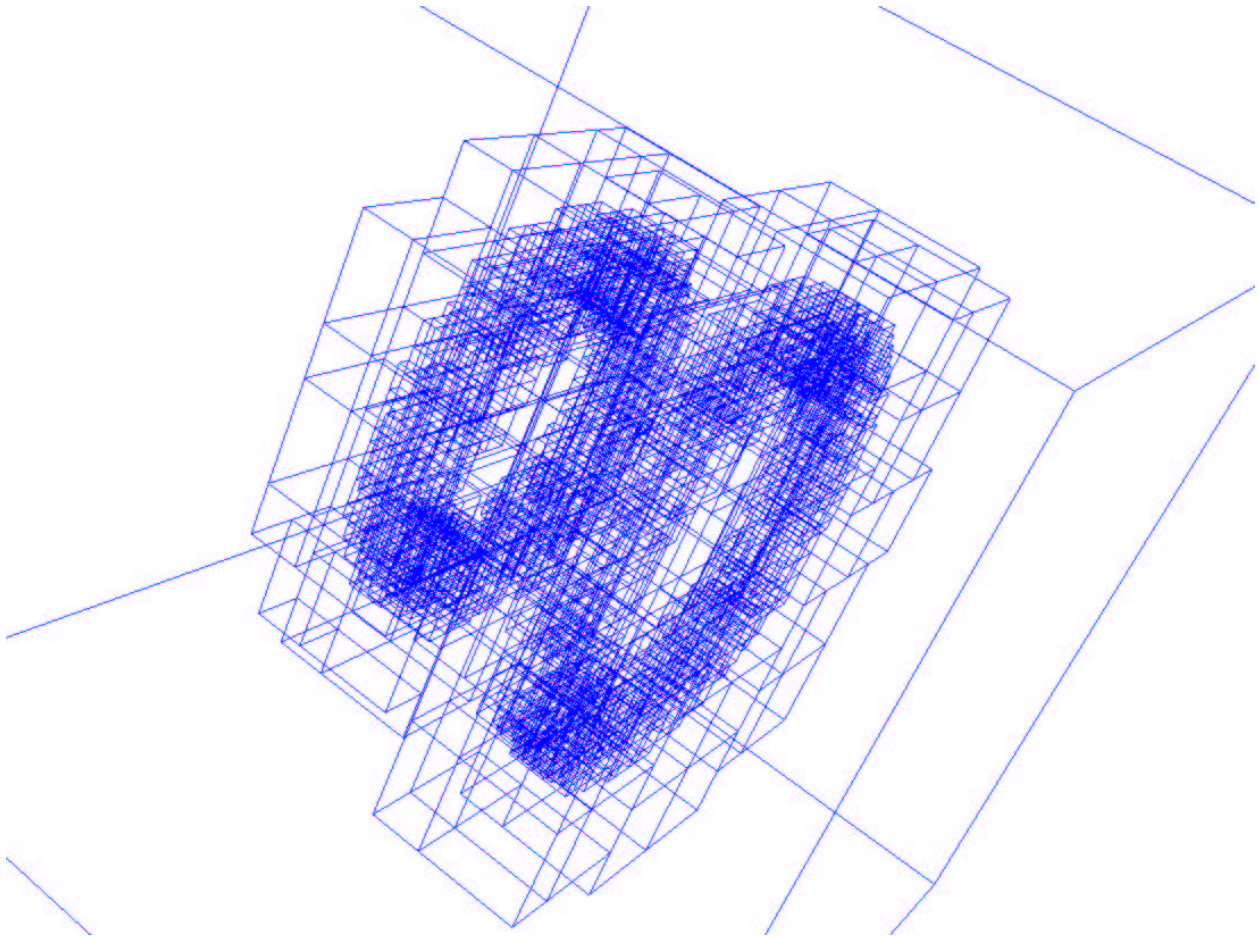
- The equation:

$$\begin{cases} \triangle u & = 0 \quad \text{on } U \\ u & = 0 \quad \text{in } \partial U \end{cases}$$

  where $u^0 = 1$.

- The grid con£guration (3D):

Level	# of Boxes	# of Points
0	1	32768
1	106	279552
2	1449	2944512



The pro£ling results on Seberg

- Overview (2.279; in seconds):

	Titanium Chombo	C++/Fortran Chombo
initialization	7.51 (7.81%)	16.31 (20.61%)
solveAMR	88.66 (92.19%)	62.82 (79.39%)
total	96.17 (100%)	79.13(100%)
# of iterations	6	6
¤ops (million)	9750	10010

- Residual (in max norm):

Iteration	Titanium Chombo	C++/Fortran Chombo
0	6144	6144
1	0.272746	0.272751
2	0.253767	0.253812
3	0.009100	0.009215
4	0.000370	0.000358
5	5.09E-06	4.78E-06
6	2.08E-07	1.63e-07

The pro£ling results on Seberg

- Details (in seconds):

Operations in solveAMR	Titanium	C++/Fortran
residual	6.65 (7.50%)	11.75 (18.70%)
AMRVCycleMG	82.01 (92.50%)	50.41 (80.25%)
residual	8.04 (9.07%)	
Update	3.11 (3.51%)	
Average	0.34 (0.38%)	
Ipwc	3.29 (3.71%)	
mgRelax	64.63 (72.90%)	37.09 (59.04%)
Update	2.10 (2.37%)	
Average	2.57 (2.90%)	
Ipwc	3.56 (4.02%)	
LevelGSRB	54.16 (61.09%)	33.43 (53.22%)
GSRB	18.12 (20.44%)	11.85 (18.86%)
Exchange	25.15 (28.37%)	15.56 (24.77%)
CFInterp	10.47 (11.81%)	5.37 (8.55%)
applyHPhyBC	0.00 (0.00%)	0.64 (1.02%)
residual	0.91 (1.03%)	
bottom solve	0.21 (0.24%)	
total	88.66 (100%)	62.82 (100%)

The pro£ling results on Seberg

- Details (in seconds):

Operation	Titanium	C++/Fortran
Exchange	30.77 (34.7%)	18.77 (29.0%)
GSRB	18.12 (20.5%)	11.85 (18.9%)
CFInterp	17.56 (19.8%)	
Re¤ux	7.70 (8.7%)	13.80 (22.0%)
Ipwc	6.85 (7.7%)	
Average	2.91 (3.3%)	
residual	2.33 (2.6%)	
total	86.24 (97.2%)	44.42 (70.71%)

The pro£ling results on Seaborg

- Overview (2.283; in seconds):

	Titanium Chombo	C++/Fortran Chombo
initialization	15.41 (6.55%)	61.00 (16.64%)
solveAMR	219.70 (93.45%)	305.33 (83.36%)
total	235.11 (100%)	366.53 (100%)
# of iterations	6	6
FP (million)	6874	8214
FMA (million)	1443	1713
total	8317(9760⋆)	9927

- Remember on Seberg:

	Titanium Chombo	C++/Fortran Chombo
initialization	7.51 (7.81%)	16.31 (20.61%)
solveAMR	88.66 (92.19%)	62.82 (79.39%)
total	96.17 (100%)	79.13(100%)
¤ops (million)	9750	10010

The pro£ling results on Seaborg

- Details (in seconds):

Operations in solveAMR	Titanium	C++/Fortran
residual	15.55 (7.08%)	34.63 (11.33%)
AMRVCycleMG	204.2 (92.94%)	269.60 (88.24%)
residual	19.77 (9.00%)	
Update	7.10 (3.23%)	
Average	0.73 (0.33%)	
Ipwc	6.39 (2.91%)	
mgRelax	164.6 (74.92%)	217.04 (71.04%)
Update	4.61 (2.10%)	
Average	3.87 (1.76%)	
Ipwc	6.05 (2.75%)	
LevelGSRB	145.1 (66.04%)	165.33 (54.11%)
GSRB	56.13 (25.55%)	106.94 (35.00%)
Exchange	60.91 (27.72%)	27.18 (8.90%)
CFInterp	27.63 (12.58%)	26.96 (8.82%)
applyHPhyBC	0.00 (0.00%)	4.07 (1.33%)
residual	2.11 (0.96%)	
bottom solve	0.50 (0.23%)	
total	219.7 (100%)	305.5 (100%)

The pro£ling results on Seaborg

- Details (in seconds):

Operation	Titanium	C++/Fortran
Exchange	73.16 (33.30%)	32.69 (10.70%)
GSRB	56.13(25.55%)	106.94 (35.00%)
CFInterp	46.05 (20.96%)	
Re¤ux	18.60 (8.47%)	43.88 (14.36%)
Ipwc	12.44 (5.66%)	
Average	4.60 (2.09.%)	
residual	4.15 (1.89%)	
total	215.13 (97.92%)	183.51 (60.06%)

- Remember on seberg:

Operation	Titanium	C++/Fortran
Exchange	30.77 (34.7%)	18.77 (29.0%)
GSRB	18.12 (20.5%)	11.85 (18.9%)
CFInterp	17.56 (19.8%)	
Re¤ux	7.70 (8.7%)	13.80 (22.0%)
Ipwc	6.85 (7.7%)	
Average	2.91 (3.3%)	
residual	2.33 (2.6%)	
total	86.24 (97.2%)	44.42 (70.71%)

The pro£ling results: summary

- Exchange is the most expensive operation in Titanium Chombo.
- Titanium Chombo has better performance portability.

Operations in solveAMR	Titanium (%)	C++/Fortran (%)
residual	-0.42	-7.37
AMRVCycleMG	0.44	7.99
residual	-0.07	
Update	-0.28	
Average	-0.05	
Ipwc	-0.8	
mgRelax	2.02	12.00
Update	-0.27	
Average	-1.14	
Ipwc	-1.27	
LevelGSRB	4.95	0.89
GSRB	5.11	16.14
Exchange	-0.65	-15.87
CFInterp	0.77	0.27
applyHPhyBC	0.00	0.31
residual	-0.07	
bottom solve	-0.01	

- GSRB is the most platform-sensitive operation in C++/Fortran Chombo.

Performance improvement and portability

1. Current localCopy method:

```
public final inline local void localCopy(
       template BoxedArray<T> local from_BA,
       RectDomain<SPACE_DIM> from_domain){

       box=from_domain*this.m_domain;
       foreach (point in box)
           this.m_array[point]=from_BA.m_array[point];
}
```

2. An alternative using Titanium-array copy method:

```
public final inline local void localCopy(
       template BoxedArray<T> local from_BA,
       RectDomain<SPACE_DIM> from_domain){

       this.m_array.copy(
            from_BA.getLocalArray().restrict(from_domain));
}
```

Performance improvement and portability

1. Exchange vs. Exchange()$\star$ (in seconds):

Operation	Seberg (2.279)	Seaborg (2.283)
Exchange	30.8	73.2
Exchange$\star$	54.2	140.3

2. Is the performance of Ti-array copy method portable?

Future work

1. Scalability study.
2. AMR in ocean modeling.