Chombo-Crunch case study

Andrey Ovsyannikov*
NESAP Postdoc, NERSC

with David Trebotich and Brian Van Straalen (CRD, LBL)

*Email: aovsyannikov@lbl.gov

November 3, 2016
Chombo-Crunch

CFD + multi-component geochemical reactive transport in very complex pore scale geometry:

\[
\begin{align*}
\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} + \nabla p &= \nu \Delta \mathbf{u}, \quad \nabla \cdot \mathbf{u} = 0 \\
\frac{\partial \rho c_k}{\partial t} + \nabla \cdot \rho \mathbf{u} c_k &= \nabla \cdot \rho D_k \nabla c_k + \rho r_k
\end{align*}
\]

- Legacy flat MPI code
- 0.6M SLOC: C++ 90%, Fortran 10%
- Dynamic local refinement (AMR)
- 2\textsuperscript{nd}-order finite-volume (low AI)
- 2\textsuperscript{nd}-order Crank-Nicolson time integration or backward Euler
- PETSC AMG linear solver
- Geometry from image data
- Geochemistry: point-by-point
- Scalable (100K+ processors)
**Embedded boundary method**

**EBIndexSpace**: geometric momentum (e.g. volume fractions, area fractions, centroids)
Stencils are computed at run time. Number of faces (“edges”) on irregular cells is not constant.

- Random memory access pattern (pure utilization of cache lines and vector units).
- Tiling is not straightforward.
**Hotspot pattern**

- 30-40% in Chombo: computing of fluxes, extrapolation, mass redistribution, normalization by VOF,...
- 30-40% in PETSC AMG (KSPSolve)
- 20-40% MPI (Waitall, Barrier)
Fraction of covered boxes depends on application: it may vary from 5% up to 70% (e.g., fractured dolomite). For porous media like shale it is typically ~30-40%.
Box pruning: Impact on performance

Reduced memory footprint and checkpoint size

Pruning can reduce a run time as well (if the same number of PE’s is used after pruning. It requires more than 1 box per PE).

<table>
<thead>
<tr>
<th># boxes</th>
<th># covered boxes</th>
<th>plotfile [GiB]</th>
<th>checkpoint [GiB]</th>
<th>memory [GiB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>32 (6.25%)</td>
<td>7.37</td>
<td>6.01</td>
<td>216.5</td>
</tr>
<tr>
<td>4096</td>
<td>512 (12.5%)</td>
<td>59</td>
<td>48.09</td>
<td>1459</td>
</tr>
<tr>
<td>32768</td>
<td>6862 (20.9%)</td>
<td>472</td>
<td>384.75</td>
<td>11211</td>
</tr>
</tbody>
</table>
Improving of load imbalance

Communication time over MPI ranks (on single HSW node)

Load balancer based on num. of cells

Feedback-based load balancer

20% decrease of total run time on Cori P1.
Squeezing memory footprint

Elimination of excessive temporary variables (e.g., conservative variables) in operator for computing advective terms:
EBAdvectLevelIntegrator class replaced EBPatchAdvect
EBAdvectPatchIntegrator class replaced EBLevelAdvect

Memory footprint: 1.5x decrease

MPI rank: 0 to 511

Footprint, MB

original

optimized

 MPI rank: 0 to 511

Footprint (MB)
Valgrind (massif tool) provides detailed output on heap usage

```
valgrind --tool=massif ./chombo.ex
ms_print massif.out > heapDump.txt
```

```
| -> 10.99% (12,007,104B) 0x7C36EC: BaseEBFaceFAB<double>::define(EBISBox const&, Box const&, int, int) (in viscoelasticDriver3d.Linux.64.g++.gfortran.OPTHIGH.PETSC.ex) |
|  | 05.94% (6,488,064B) 0x7C15A7: EBFaceFAB::define(EBISBox const&, Box const&, int, int) (in viscoelasticDriver3d.Linux.64.g++.gfortran.OPTHIGH.PETSC.ex) |
|  | 04.45% (4,866,048B) 0x5F4E15: EBFaceFAB::define(EBISBox const&, Box const&, int, int) (in viscoelasticDriver3d.Linux.64.g++.gfortran.OPTHIGH.PETSC.ex) |
|  | 04.45% (4,866,048B) 0x5C30C4: EBFaceFAB::define(EBISBox const&, Box const&, int, int) (in viscoelasticDriver3d.Linux.64.g++.gfortran.OPTHIGH.PETSC.ex) |
|  | 31.18% (34,078,720B) 0x47303C: BaseFab<double*>::resize(Box const&, int, double**) (in viscoelasticDriver3d.Linux.64.g++.gfortran.OPTHIGH.PETSC.ex) |
|  | 31.18% (34,078,720B) 0x4A9205: BaseIVFAB<double>::define(IntVectSet const&, EBGraph const&, int const&) (in viscoelasticDriver3d.Linux.64.g++.gfortran.OPTHIGH.PETSC.ex) |
|  | 04.32% (4,718,592B) 0x5F4E95: EBFaceFAB::define(EBISBox const&, Box const&, int, int) (in viscoelasticDriver3d.Linux.64.g++.gfortran.OPTHIGH.PETSC.ex) |
```
From flat MPI to MPI+OpenMP

16 Boxes, 16 MPI ranks

<table>
<thead>
<tr>
<th>rank 0</th>
<th>rank 1</th>
<th>rank 2</th>
<th>rank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank 4</td>
<td>rank 5</td>
<td>rank 6</td>
<td>rank 7</td>
</tr>
<tr>
<td>rank 8</td>
<td>rank 9</td>
<td>rank 10</td>
<td>rank 11</td>
</tr>
<tr>
<td>rank 12</td>
<td>rank 13</td>
<td>rank 14</td>
<td>rank 15</td>
</tr>
</tbody>
</table>

16 Boxes, 4 MPI ranks, 4 threads per rank

```c
for (DataIterator dit = a_grids.dataIterator(); dit.ok(); ++dit) {
    averageCellToFace(a_fluxData[dit()][idir], a_cellData[dit()], a_grids[dit()], a_ebisl[dit](), ...);
}
```

```c
#pragma omp parallel for
for (int mybox = 0; mybox < nbox; mybox++) {
    averageCellToFace(a_fluxData[dit][mybox][idir], a_cellData[dit][mybox], a_grids[dit][mybox], a_ebisl[dit](), ...);
}
```
Fixing data races with Intel Inspector

Detected all race conditions. Introduced `omp atomic`, critical and `threadprivate` to fix data races.
Further footprint reduction due to shared memory

1.7x less memory for MPI+OpenMP: for 6 spheres benchmark memory drops from 2880 MB/node to 1672 Mbytes/node.

Performance of MPI+OpenMP has not been assessed. Issues with dead locks.
Overall performance speedup

**Profile**

**CPU**
- Computation: 40.57%

**Memory Utilization**
- Cache hit ratio: 99.2% hits
- Miss HiMem (MBytes): 462.5 ST per TLB miss: 2918.53 ref

**Function/Region Profile**
- 52.1% = MPI Waitall
- 6.2% = AMRINSGeometry
- 4.4% = EBarith...tonePath

**Load Imbalance**
- Programming Model: 59.41%

Data Movement: No data collected.

---

**Profile**

**CPU**
- Computation: 64.10%

**Memory Utilization**
- Cache hit ratio: 98.4% hits
- Miss HiMem (MBytes): 339.3 ST per TLB miss: 2123.02 ref

**Function/Region Profile**
- 21.4% = MPI Waitall
- 9.5% = AMRINSGeometry
- 6.9% = EBarith...tonePath

**Load Imbalance**
- Programming Model: 35.85%

Data Movement: No data collected.
Overall performance speedup

- 248 sec
- 24,788 samples

- 164 sec
- 16,398 samples

MPI_Waitall (52.5%)
Samples: 13,011
Imbalance: 30.3%
Imb Samples: 5650.4
Relative KNL-to-HSW performance strongly depends on complexity of geometry (i.e. fraction of irregular cells):

<table>
<thead>
<tr>
<th>benchmark</th>
<th>GFLOPS/sec (HSW)</th>
<th>GFLOPS/sec (KNL)</th>
<th>KNL-to-HSW</th>
</tr>
</thead>
<tbody>
<tr>
<td>100% regular (AMR Poisson)</td>
<td>20.7</td>
<td>30.8</td>
<td>1.49</td>
</tr>
<tr>
<td>1 sphere (0.5% irregular cells)</td>
<td>12.8</td>
<td>10.5</td>
<td>0.82</td>
</tr>
<tr>
<td>6 spheres (5% irregular cells)</td>
<td>7.5</td>
<td>4.4</td>
<td>0.58</td>
</tr>
<tr>
<td>200 spheres (15% irregular cells)</td>
<td>4.0</td>
<td>1.6</td>
<td>0.40</td>
</tr>
</tbody>
</table>
Relative KNL-to-HSW performance strongly depends on problem size:

<table>
<thead>
<tr>
<th>Mesh size</th>
<th>GFLOPS/sec (HSW)</th>
<th>GFLOPS/sec (KNL, HBM)</th>
<th>GFLOPS/sec (KNL, DDR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$64^3$</td>
<td>7.5</td>
<td>1.6</td>
<td>1.4</td>
</tr>
<tr>
<td>$128^3$</td>
<td>20.7</td>
<td>7.3</td>
<td>6.3</td>
</tr>
<tr>
<td>$256^3$</td>
<td>17.1</td>
<td>19.4</td>
<td>10.2</td>
</tr>
<tr>
<td>$512^3$</td>
<td>17.8</td>
<td>30.8</td>
<td>10.3</td>
</tr>
</tbody>
</table>
KNL, QuadFlat (benchmark fits to HBM)

Only ~1.5x speedup using HBM vs DDR!

Kernels have different performance limitation: PETSc AMG is DRAM-bandwidth bound; EBChombo kernels (e.g., AggStencil::apply) are latency bound. Detailed roofline assessment of all major hotspots is in progress.
KNL: Multi node strong scaling study

KNL: 1 to 16 nodes on Gerty (64 MPI ranks per node, 4 remaining cores for OS)
Ivy Bridge: 1 to 16 nodes on Edison (24 MPI ranks per node)

Solution time per timestep (sec)

Number of nodes

Chombo-Crunch: Strong scaling

KNL (DDR)

IvyBridge

Ideal scaling
Summary

- Reduced commun. time (MPI_Waitall) → 20-30% speedup.
- BetterEB (AggStencil) → leads to another 20% speedup.
- 1.5x reduction of memory footprint due to optimization of advection terms. 1.7x reduction due to threading.
- MPI+OpenMP development: fixed race conditions AMG solver (PETSC) remains unthreaded.
  - Short-term: 16 MPI ranks + 4 threads in Chombo;
    16 MPI ranks for PETSc.
  - Long-term: MPI Communication Endpoints.
- KNL-to-Haswell (single node) performance: strongly depends on geometry (fraction of irregular cells) and problem size: from 1.5x to 0.2x.
- Vectorization is pure: only 5-10% of speedup by using AVX512 vector instructions.
- Started to work on the kernel (AggStencil) to improve data locality in irregular part of computation.
Thank you. Questions?