Various Implementations of FMM and their performance on future architectures

Rio Yokota

Multiresolution Interactions Workshop
August 28-29 (2015)
Duke University, Durham, NC
Advancement of FMM over the years
- each FMM code contains only a small subset of these inventions -
Various Implementations
- implementation matters -

<table>
<thead>
<tr>
<th>Language</th>
<th>exaFMM</th>
<th>scalFMM</th>
<th>falcON</th>
<th>scafacos</th>
<th>pKIFMM</th>
<th>PEPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction list</td>
<td>C++</td>
<td>C++11</td>
<td>C++</td>
<td>C, F90</td>
<td>C++</td>
<td>Fortran 2003</td>
</tr>
<tr>
<td>Series expansion</td>
<td>DTT</td>
<td>2-list</td>
<td>DTT</td>
<td>Spherical + Chebyshev</td>
<td>Equivalent charges</td>
<td>Barnes-Hut</td>
</tr>
<tr>
<td>Accuracy control</td>
<td>varying p &amp; θ</td>
<td>constant p</td>
<td>varying θ</td>
<td>varying p</td>
<td>constant p</td>
<td>constant θ</td>
</tr>
<tr>
<td>MPI</td>
<td>o</td>
<td>o</td>
<td>x</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Threading model</td>
<td>TBB+OpenMP</td>
<td>OpenMP</td>
<td>OpenMP</td>
<td>OpenMP</td>
<td>OpenMP</td>
<td>pthreads</td>
</tr>
<tr>
<td>SIMD</td>
<td>AVX</td>
<td>AVX</td>
<td>N/A</td>
<td>N/A</td>
<td>AVX</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Modular view of FMM codes
- each of us should do what we do best -

exaFMM has
1. Kernels (1444 lines)
2. Local tree (589 lines)
3. Lists (344 lines)
4. Partition (245 lines)
5. Global tree (447 lines)

Reuse & swap among FMM codes

Separation of concerns

Kernels

Common interface

Tree

MPI part
Various Kernels
- the compute-memory tradeoff -

<table>
<thead>
<tr>
<th>Type of expansion (+M2L acceleration)</th>
<th>Storage</th>
<th>Arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartesian Taylor</td>
<td>$O(p^3)$</td>
<td>$O(p^6)$</td>
</tr>
<tr>
<td>Cartesian Chebychev</td>
<td>$O(p^3)$</td>
<td>$O(p^6)$</td>
</tr>
<tr>
<td>Spherical harmonics</td>
<td>$O(p^2)$</td>
<td>$O(p^4)$</td>
</tr>
<tr>
<td>Spherical harmonics+rotation</td>
<td>$O(p^2)$</td>
<td>$O(p^3)$</td>
</tr>
<tr>
<td>Spherical harmonics+FFT</td>
<td>$O(p^2)$</td>
<td>$O(p^3 \log^2 p)$</td>
</tr>
<tr>
<td>Planewave</td>
<td>$O(p^3)$</td>
<td>$O(p^3)$</td>
</tr>
<tr>
<td>Equivalent charges</td>
<td>$O(p^3)$</td>
<td>$O(p^4)$</td>
</tr>
<tr>
<td>Equivalent charges+FFT</td>
<td>$O(p^3)$</td>
<td>$O(p^3 \log p)$</td>
</tr>
</tbody>
</table>

Memory access (Expensive) | Algebraic | Sampling | Use of symmetry | Geometric | Compute (Cheap)

**M2L kernel**

![Graph showing time per cell pair vs order of expansion](image)

- Spherical rotation: $O(p^{2.5})$
- Cartesian Taylor: $O(p^{4.5})$

**Double precision performance (Gflop/s)**

- Intel Sandy Bridge
- AMD Abu Dhabi
- IBM BG/Q
- Fujitsu FX10
- NVIDIA Kepler
- Intel Xeon Phi
Various Interaction Lists
- finding the well-separated boxes -

Carrier et al. (1988)

Gumerov & Duraiswami (2008)
Andoh et al. (2012)
Kurzak & Pettitt (2006)
Gumerov & Duraiswami (2008)
Dual Tree Traversal
- a fast algorithm for finding well-separated boxes -


```c
void traverse(A, B) { 
    if (MAC(A, B)) { // can approximate and we are done
        approximate(A, B);
    } else if (leaf(A) && leaf(B)) {
        direct(A, B);
    } else if (leaf(A)) { // A is leaf but B is not
        for (b : B’s children) traverse(A, b);
    } else if (leaf(B)) { // B is leaf but A is not
        for (a : A’s children) traverse(a, B);
    } else if (A == B) { 
        for ((a, b) : a pair of A’s child s.t. a ≤ b) 
            traverse(a, b);
    } else if (radius(A) < radius(B)) { 
        // descend along the larger one (B)
        for (b : B’s children) traverse(A, b);
    } else { 
        for (a : A’s children) traverse(a, B);
    }
}
```
Dual Tree Traversal
- a fast algorithm for finding well-separated boxes -

- The definition of well-separatedness (size of neighbor region) can be adjusted flexibly without modifying the code

- It is applicable to **adaptive trees** without any modification

- It lends itself to **MPI parallelization** without any modification (by simply using the local essential tree as the source tree)

- It can handle **mutual M2L interaction**, and can satisfy Newtons third law (M2L is neither target centric nor source centric, but completely symmetric)

- It works well with **task based threading** tools like Intel TBB, Cilk, etc., where tasks are spawned while the tree is traversed

- The cells **don't have to be cubic**. For example, high aspect ratio rectangles or hierarchical K-means is permitted.

- It can be implemented in **less than 100 lines of code**, and is therefore trivial to debug
h-p FMM
- how to achieve a given accuracy with minimum work -

h: well separatedness (admissibility)
p: order of expansion (rank)

Spatially varying rank
fine grain load imbalance

Spatially varying admissibility

SIMD friendly
Domain Decomposition
- balancing the workload while minimizing communication -
Interaction list bloat
- trading code bloat with interaction list bloat? -

Morton HOT

New ORB
Communication Complexity of FMM  
- aiming for a better asymptotic limit for scalability -

**PROVABLY GOOD PARTITIONING AND LOAD BALANCING ALGORITHMS FOR PARALLEL ADAPTIVE N-BODY SIMULATION**

SHANG-HUA TENG

A massively parallel adaptive fast-multipole method on heterogeneous architectures

Ilya Lashuk, Aparna Chandramowlishwaran, Harper Langston, Tuan-Anh Nguyen, Rahul Sampath, Aashay Shringarpure, Richard Vuduc, Lexing Ying, Denis Zorin, and George Biros

Communication Complexity of the Fast Multipole Method and its Algebraic Variants

Rio Yokota, George Turkiyyah, David Keyes

\[
\log P - 1 \sum_i 2^i = \mathcal{O}(P)
\]

\[
\log P - 1 \sum_i \min(2^{\log P - i - 1}, 2^i) = \mathcal{O}(\sqrt{P})
\]

force 2:1 refinement ratio

**Table:**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Processes</th>
<th>Data per Process</th>
<th>Communication complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teng (1998)</td>
<td>(\mathcal{O}(P))</td>
<td>(\mathcal{O}\left((N/P)^{2/3}(\log N + \mu)^{1/3}\right))</td>
<td>(\mathcal{O}\left(P(N/P)^{2/3}(\log N + \mu)^{1/3}\right))</td>
</tr>
<tr>
<td>Lashuk <em>et al.</em> (2009)</td>
<td>(\mathcal{O}(\sqrt{P}))</td>
<td>(\mathcal{O}\left((N/P)^{2/3}\right))</td>
<td>(\mathcal{O}\left(\sqrt{P}(N/P)^{2/3}\right))</td>
</tr>
<tr>
<td>Yokota <em>et al.</em> (2014)</td>
<td>Global (\mathcal{O}(\log P)) Local (\mathcal{O}(1))</td>
<td>Global (\mathcal{O}(1)) Local (\mathcal{O}\left((N/P)^{2/3}\right))</td>
<td>Global + Local (\mathcal{O}\left(\log P + (N/P)^{2/3}\right))</td>
</tr>
</tbody>
</table>

**Notes:**
P: # of processes, N: global problem size

**Formulas:***

- \(\mathcal{O}\) notation for Big O notation
- \(\log\) denotes logarithm base 2
- \(\min\) function to take the minimum of two values
- \(\mathcal{O}(\sqrt{P})\) indicates a complexity proportional to the square root of the number of processes

**Diagram:**

- Hypercube alltoall operation
- 2:1 refinement ratio

**Graphs:**

- Uniform load distribution
- Nonuniform load distribution
- Pathological load distribution

**References:**

- Aims for better asymptotic limit for scalability

**Context:**

- Fast Multipole Method (FMM) is a technique for efficiently approximating the long-range interactions in particle simulations
- MPI (Message Passing Interface) part of the document
- Focus on communication complexity and load balancing strategies for parallel computing

**Visual Elements:**

- Diagrams illustrating load distribution
- Mathematical expressions for communication complexity
- Table summarizing process and data complexities for different studies
Communication Pattern of FMM
- how to avoid an all-to-all communication -
Performance Model for Communication
- understanding the asymptotic constants -

\[
\begin{align*}
T_{\alpha-\beta-\gamma} &= \alpha + n\beta + (h - h_m)\gamma \\
T_{\alpha-\text{penalty}} &= c\alpha + n\beta + (h - h_m)\gamma \\
T_{\beta-\text{penalty}} &= \alpha + n\beta \frac{B_{\text{max}}}{B} + (h - h_m)\gamma \\
T_{\alpha-\text{penalty}} &= \alpha + n\beta + c(h - h_m)\gamma
\end{align*}
\]

\(\alpha\): latency
\(\beta\): inverse bandwidth
\(\gamma\): delay per hop
\(n\): message size
\(h\): number of hops
\(h_m\): minimum of \(h\)
\(c\): number of cores
\(B\): bandwidth
\(B_{\text{max}}\): maximum bandwidth

Shaheen2 (Cray XC40)
Mira (BG/Q)
Titan (Cray XK7)
Communication Complexity $\mathcal{O}(\log P + (N/P)^{2/3})$

Machine: Shaheen2 (Cray XC40)
Kernel: Laplace (Cartesian expansion)
Distribution: Random in cube
Partition: Hashed Octree
MPI: isend,irecv (with overlap)

Strong scaling (N=300,000,000)
Weak scaling (N=100,000,000 per node)
Next steps
- what needs to happen -

1. Stable interface between the modules
2. Standard benchmark for each module
3. Sharing highly optimized implementations
4. Software engineering tools