Task-based Parallelization of the Fast Multipole Method on NVIDIA GPUs and Multicore Processors

Emmanuel Agullo, a Bérenger Bramas, a Olivier Coulaud, a Eric Darve, b Matthias Messner, a Toru Takahashi c

a INRIA Bordeaux – Sud-Ouest, France
b Institute for Computational and Mathematical Engineering, Stanford University
c Department of Mechanical Science and Engineering, Nagoya University

March 21 2013
Outline

1. Fast multipole method
2. Task-based parallelization
3. Numerical benchmarks on multicore and NVIDIA GPUs
Fast Multipole Method

Applications:

- Gravitational simulations, molecular dynamics, electrostatic forces
- Interpolation using radial basis functions: meshless methods, graphics (data smoothing)
- Imaging and inverse problems: subsurface characterization and monitoring, imaging
- $O(N)$ fast linear solvers: LU factorization, matrix inverse, for dense and sparse matrices, “FastLAPACK”
Matrix-vector products

FMM: a method to calculate

\[ \phi(x_i) = \sum_{j=1}^{N} K(x_i, x_j) \sigma_j, \quad 1 \leq i \leq N \]

in \( \mathcal{O}(N) \) or \( \mathcal{O}(N \ln^{\alpha} N) \) arithmetic operations.

There are many different FMM formulations.

Most of them rely on **low-rank** approximations of the kernel function \( K \) in the form:

\[ K(x, y) = \sum_{q=1}^{p} u_q(x) \sum_{s=1}^{p} T_{qs} v_s(y) + \varepsilon \]
Many formulas are possible to obtain a low-rank approximation.

We use an approach based on interpolation formulas (Chebyshev polynomials) for smooth functions. This allows building an FMM that is:

- **Very general:** any smooth (even oscillatory) kernel can be used.
- **Simple** to apply: we only need a routine that evaluates $K$.
- With SVD acceleration, has **optimal rank**.

In some cases, it may be less efficient than specialized methods for $K(x, y) = 1/|x - y|$.  

Convergence of Chebyshev interpolation with SVD acceleration

Error vs SVD cutoff for $K = 1/r$

2D prolate mesh

Error vs target accuracy, $K = e^{ikr}/r$

Task-based parallelization

<table>
<thead>
<tr>
<th>Task</th>
<th>Concurrency</th>
<th>Load-balancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2L (lower level)</td>
<td>High</td>
<td>Homogeneous</td>
</tr>
<tr>
<td>M2L (higher level)</td>
<td>Low</td>
<td>Homogeneous</td>
</tr>
<tr>
<td>P2P</td>
<td>High</td>
<td><strong>Heterogeneous</strong></td>
</tr>
<tr>
<td>M2M/L2L</td>
<td>Low</td>
<td>Homogeneous</td>
</tr>
</tbody>
</table>
Multicore parallelization with OpenMP: fork-join

function FMM(tree)

    // Near-field
    P2P(tree.levels[tree.height-1]);

    // Far-field
    P2M(tree.levels[tree.height-1]);

    forall the level l from tree.height-2 to 2 do
        M2M(tree.levels[l]);

    forall the level l from 2 to tree.height-2 do
        M2L(tree.levels[l]);
        L2L(tree.levels[l]);

    M2L(tree.levels[tree.height-1]);
    L2P(tree.levels[tree.height-1]);
Parallel for loops

The loop over cells at a given level are parallelized:

function M2L(level)

#pragma omp parallel for
defined

cell cl in level.cells do
kernel.m2l(cl.local,
    cl.far_field.multipole);

Agullo, Bramas, Coulaud, Darve, Messner, Takahashi
Improving the scalability

- The top of the tree represents a parallel bottleneck.
- P2P tasks can be highly heterogeneous.
- Solution:
  - mix heavily parallel sections with more sequential ones.
  - Try to execute large tasks first and finish with small tasks.
  - On heterogeneous platforms, map tasks to appropriate cores.
Interleaving the far field and near field calculations

```c
#pragma omp parallel
#pragma omp sections
#pragma omp section
  P2Ptask(tree.levels[tree.height-1])
#pragma omp sections
  P2Mtask(tree.levels[tree.height-1])
    forall the level l from tree.height-2 to 2 do
      M2Mtask(tree.levels[l])
      forall the level l from 2 to tree.height-2 do
        M2Ltask(tree.levels[l])
        L2Ltask(tree.levels[l])
        M2Ltask(tree.levels[tree.height-1])
#pragma omp single
  L2Ptask(tree.levels[tree.height-1])
```

Agullo, Bramas, Coulaud, Darve, Messner, Takahashi

FMM on StarPU
StarPU

- **StarPU** is a parallel runtime system for heterogeneous platforms (multicore, GPU)
- API allows expressing the DAG (directed acyclic graph) of tasks
- Runtime scheduling uses a variety of schedulers, including user-defined schedulers
- Abstract the processor architecture

Example of a task insertion using StarPU:

```c
insertTask( codelet, ACCESS_MODE, handle, ... )
```

- `codelet`: structure containing function pointers
- `ACCESS_MODE ← { READ, WRITE, READ/WRITE }`
- `handle`: StarPU data interface
- `ACCESS_MODE` and order of task insertion implicitly define the DAG.

```c
forall the level l from 2 to tree.height-2 do
    foreach block bl in tree.levels[l].blocks do
        insertTask( kernel.m2l, READ, bl.Ilist, WRITE, bl.local);
```
Priority scheduler

Task legend:
? : Unknown status
X : Not ready
R : Ready
P : Currently processed
F : Finished

Release dependencies

P2M > M2M > L2L(\text{l}) > P2P (large) > M2L(\text{l}) > P2P (small) > L2P

Agullo, Bramas, Coulaud, Darve, Messner, Takahashi
Heterogeneous priority scheduler

<table>
<thead>
<tr>
<th>Operators</th>
<th>Homogeneous CPU</th>
<th>Heterogeneous CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2M</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>M2M</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>L2L</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>P2P (Large)</td>
<td>3</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>M2L</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>P2P (Small)</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>L2P</td>
<td>6</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>P2P Restore</td>
<td>7</td>
<td>6</td>
<td>-</td>
</tr>
</tbody>
</table>

GPUs get assigned **P2P** and **M2L** operators, with a higher priority on P2P since GPUs are most efficient at that task.
Performance of OpenMP and StarPU on multicore

<table>
<thead>
<tr>
<th>Number of Particles</th>
<th>Performance Gflop/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq. fork-join × 160</td>
<td>Fork-join 160 CPUs</td>
</tr>
<tr>
<td>Seq. block FJ × 160</td>
<td>Block FJ 160 CPUs</td>
</tr>
<tr>
<td>Seq. interleaved × 160</td>
<td>Interleaved 160 CPUs</td>
</tr>
<tr>
<td>Seq. StarPU × 160</td>
<td>StarPU 160 CPUs</td>
</tr>
</tbody>
</table>

h = 8
h = 7
h = 6
h = 5

Performance Gflop/s

Number of Particles

Logarithmic scale
StarPU on heterogeneous parallel platform: execution trace

Top: Nehalem-Fermi M2070; execution time 12.5 sec.
Bottom: Nehalem-Fermi M2090; execution time 10.9 sec.

$N = 30 \times 10^6$. The 9 first lanes represent CPU cores occupancy and the 3 last ones GPUs.
Heterogeneous execution of an imbalanced FMM

Top: M2L heavy; Bottom: P2P heavy
Scheduling with empirical execution model

Scheduler has the option of using a model of the execution time to estimate the best mapping of the DAG onto resources.
Modern heterogeneous computing platforms lead to hard to schedule parallel DAGs.

Approach like OpenMP and OpenACC may prove insufficient.

Runtime optimization of parallel mapping was demonstrated.

Significant improvement in parallel scalability observed using StarPU.

Code is hardware “independent.” Scheduler determines the best core to execute a given task.