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
- **Integral equations** for Laplace, Poisson, Stokes, Navier. Both fluid and solid mechanics.
- 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

$$\phi(\mathbf{x}_i) = \sum_{j=1}^{N} K(\mathbf{x}_i, \mathbf{x}_j)\,\sigma_j, \quad 1 \leq i \leq N$$

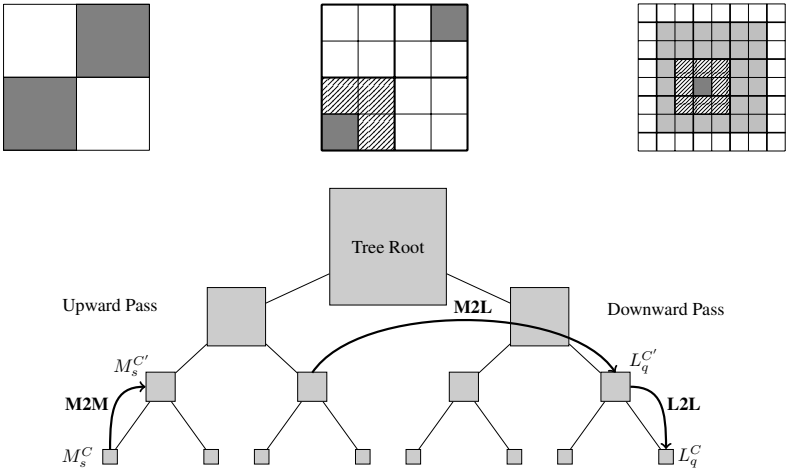
in $\mathcal{O}(N)$ or $\mathcal{O}(N\,\ln^{\alpha} N)$ arithmetic operations.

There are many different FMM formulations.

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

$$K(\mathbf{x}, \mathbf{y}) = \sum_{q=1}^{p} u_q(\mathbf{x}) \sum_{s=1}^{p} T_{qs}\, v_s(\mathbf{y}) + \varepsilon$$

Data structures and well-separated clusters



E. Darve, C. Cecka, and T. Takahashi. The fast multipole method on parallel clusters, multicore processors, and graphics processing units. *Comptes Rendus Mécanique*, 339(2–3):185–193, 2011.

Interpolatory FMM

Many formulas are possible to obtain a low-rank approximation.

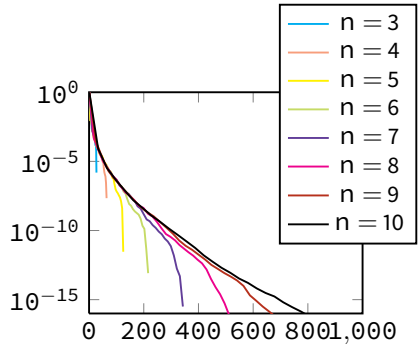
We use an approach based on **interpolation formulas** (Chebyshev polynomials) for smooth functions. This allows building an FMM that is:

- 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**.

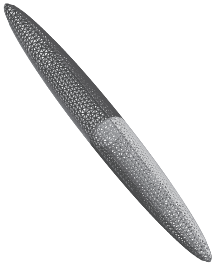
In some cases, it may be less efficient than specialized methods for $\mathrm{K}(\mathbf{x}, \mathbf{y}) = 1/|\mathbf{x} - \mathbf{y}|$.

W. Fong, E. Darve, The black-box fast multipole method, *J. Comput. Phys.*, 228(23):8712–8725, 2009.

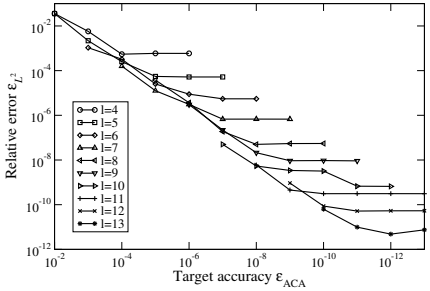
Convergence of Chebyshev interpolation with SVD acceleration



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



2D prolate mesh

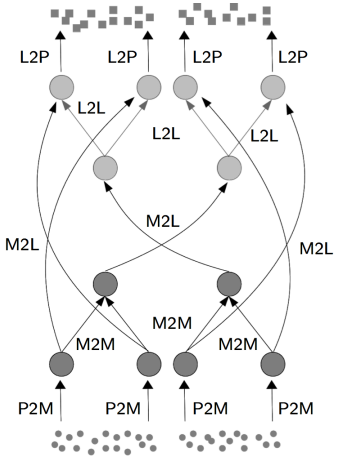


Error vs target accuracy, $K = e^{ikr}/r$

M. Messner, M. Schanz, and E. Darve. Fast directional multilevel summation for oscillatory kernels based on Chebyshev interpolation. *J. Comp. Phys.*, 2011.

Task-based parallelization

Task	Concurrency	Load-balancing
M2L (lower level)	High	Homogeneous
M2L (higher level)	**Low**	Homogeneous
P2P	High	**Heterogeneous**
M2M/L2L	**Low**	Homogeneous



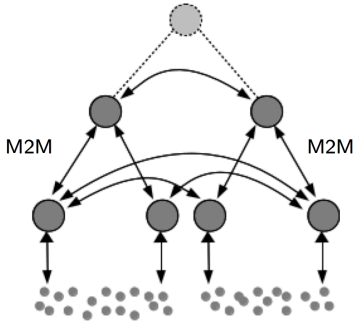
Multicore parallelization with OpenMP: fork-join

```
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

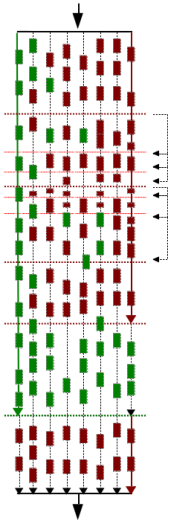
The loop over cells **at a given level** are parallelized:

```
function M2L(level)
    #pragma omp parallel for
    foreach cell cl in level.cells do
        kernel.m2l(cl.local,
                   cl.far_field.multipole);
```



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

```
#pragma omp parallel
    #pragma omp sections
        #pragma omp section
            P2Ptask(tree.levels[tree.height-1])
        #pragma omp section
            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])
```

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

E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, T. Takahashi. “Pipelining the Fast Multipole Method over a Runtime System.” arXiv preprint arXiv:1206.0115, 2012.

API Application Programming Interface

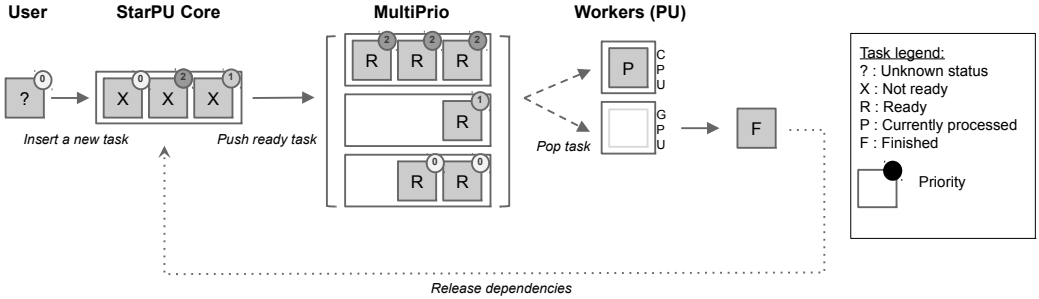
Example of a task insertion using StarPU:

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

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

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



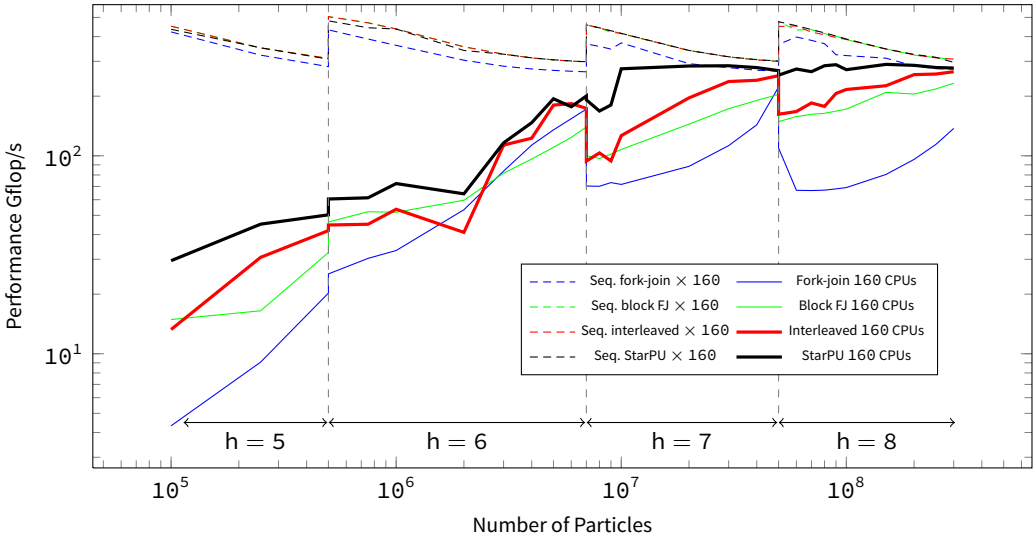
P2M > M2M > L2L(l) > P2P (large) > M2L(l) > P2P (small) > L2P

Heterogeneous priority scheduler

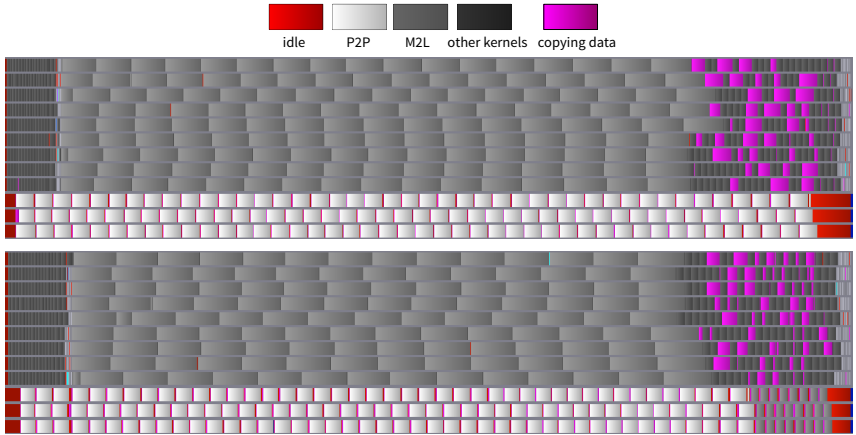
Operators	Homogeneous CPU	Heterogeneous CPU	Heterogeneous GPU
P2M	0	0	-
M2M	1	1	-
L2L	2	2	-
P2P (Large)	3	7	0
M2L	4	3	2
P2P (Small)	5	4	1
L2P	6	5	-
P2P Restore	7	6	-

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



StarPU on heterogeneous parallel platform: execution trace

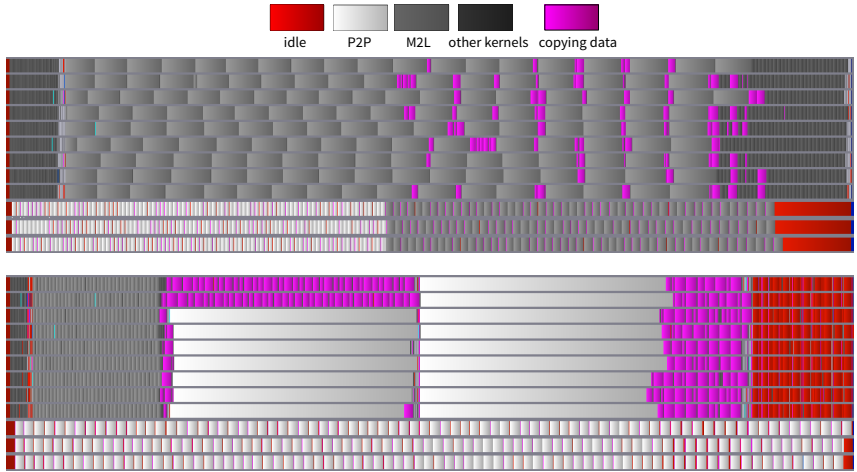


Top: Nehalem-Fermi M2070; execution time **12.5** sec.

Bottom: Nehalem-Fermi M2090; execution time **10.9** sec.

$N = 30\ 10^6$. The 9 first lanes represent CPU cores occupancy and the 3 last ones GPUs.

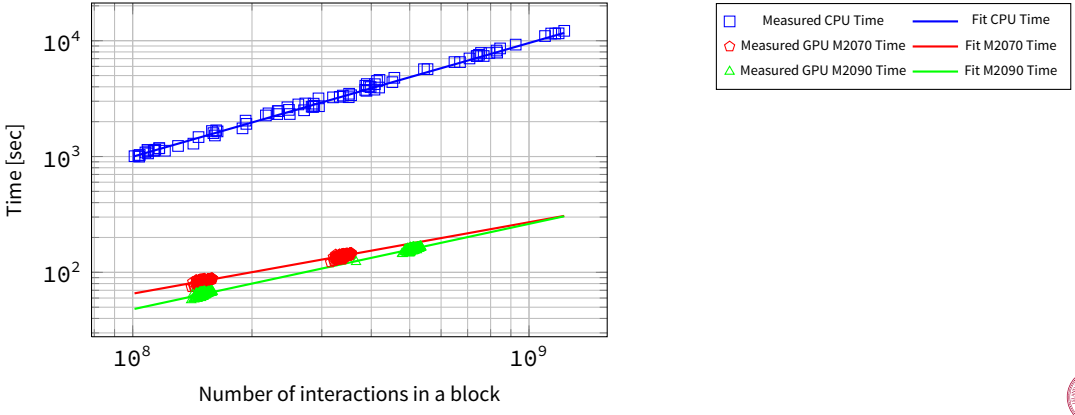
Heterogeneous execution of an imbalanced FMM



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.



Conclusion

- 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.