FMM goes GPU

A smooth trip or bumpy ride?

March 18, 2015 | B. Kohnke, I.Kabadshow | MPI BPC Göttingen & Jülich Supercomputing Centre

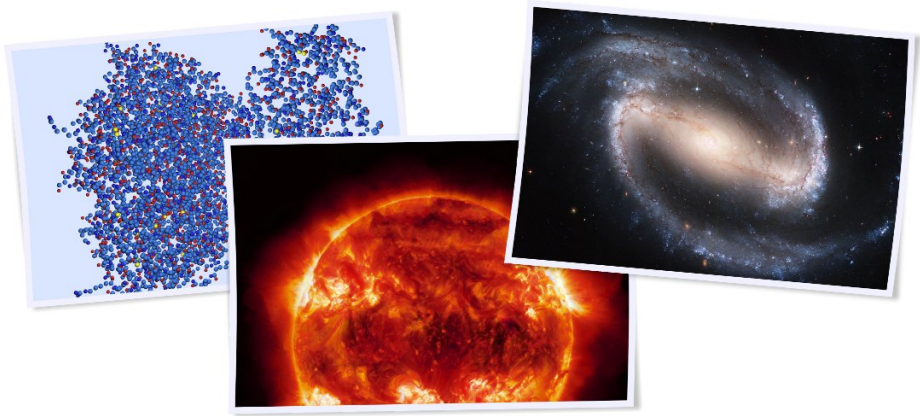
FMM goes GPU

A smooth trip or bumpy ride?

- The Fast Multipole Method
  - FMM Preliminaries
  - Operators
  - Algorithm
- GPU Parallelization Strategies
  - Naïve parallelism
  - Dynamic parallelism
- Results & Conclusions

FMM Applications

$1/r$ Long-Range Interactions in $\mathcal{O}(N)$ instead of $\mathcal{O}(N^2)$



Molecular Dynamics Plasma Physics Astrophysics

FMM Preliminaries

Different Approaches to Solve N-body Problem

Solving the N-body Problem

- Classical Approach
  - Simple load balancing
  - Computational expensive $\mathcal{O}(N^2)$
  - Can not be applied to large particle systems
- PME (Particle Mesh Ewald)
  - Computational complexity $\mathcal{O}(N \log N)$
  - FFT–based
  - Not flexible enough for our use-case
- Fast Multipole Method
  - Computational complexity $\mathcal{O}(N)$
  - Tree-based approach

N-Body Problem

Classical Approach

- For all particles compute force acting on q_i due to q_j
- Integrate equations of motion
- Determine new system configuration

$$E_c = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{q_i \cdot q_j}{r_{ij}} \tag{1}$$

N-Body Problem

Classical Approach

- For all particles compute force acting on q_i due to q_j
- Integrate equations of motion
- Determine new system configuration

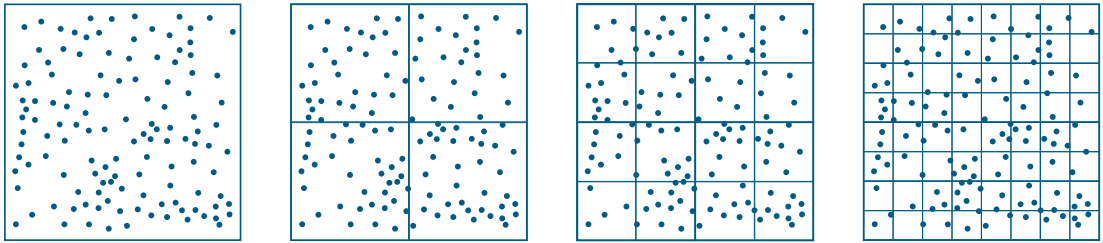
$$E_c = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{q_i \cdot q_j}{r_{ij}} \tag{1}$$

FMM on one line

$$E_C \approx \sum_{\text{level}} \sum_{\text{A}} \sum_{\text{B(A)}} \sum_{l=0}^{p} \sum_{m=-l}^{l} \sum_{j=0}^{p} \sum_{k=-j}^{j} (-1)^j \omega_{lm}^{\text{A}}(\mathbf{a}) M2L(\mathbf{R}) \omega_{jk}^{\text{B}}(\mathbf{b})$$

FMM Parameter I:

Depth of the FMM Tree d

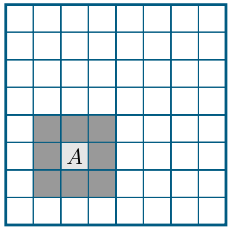


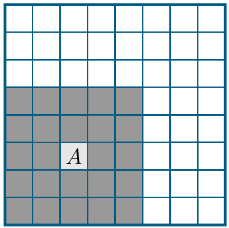
Tree Depth d, Level $L = d + 1$

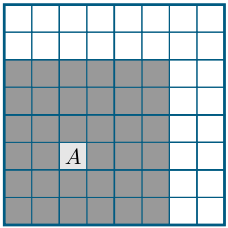
Simulation box divided into 8^d subboxes (in 3D)

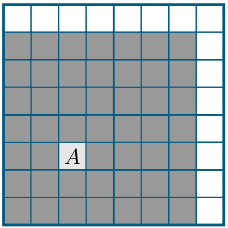
FMM Parameter II

Separation Criterion *ws*









Separation Criterion *ws*

Near field contains $(2 \cdot ws + 1)^3$ boxes

FMM Parameter III

Number of Poles p

Infinite Expansion

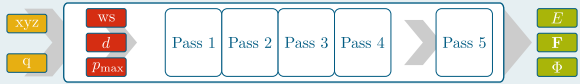
$$\frac{1}{d} = \frac{1}{|\mathbf{r} - \mathbf{a}|} = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} O_{lm}(\mathbf{a}) \cdot M_{lm}(\mathbf{r})$$

Finite Expansion Up To Order p

$$\frac{1}{d} = \frac{1}{|\mathbf{r} - \mathbf{a}|} \approx \sum_{l=0}^{p} \sum_{m=-l}^{l} O_{lm}(\mathbf{a}) \cdot M_{lm}(\mathbf{r})$$

FMM Workflow

FMM Workflow

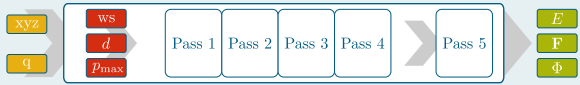


Preprocessing Steps

- Define FMM parameter ws,d,p

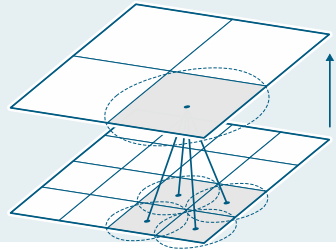
FMM Workflow

FMM Workflow



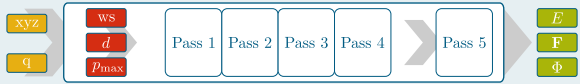
Pass 1

- Setup FMM tree and expand multipoles (P2M)
- Shift multipoles to root node (M2M)



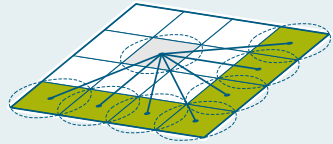
FMM Workflow

FMM Workflow



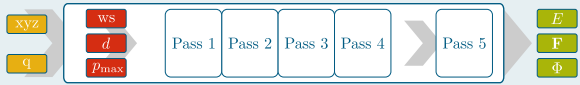
Pass 2

- Translate multipole moments into Taylor moments (M2L)



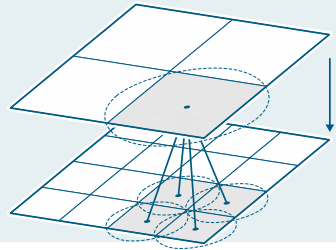
FMM Workflow

FMM Workflow



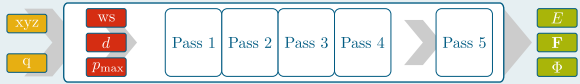
Pass 3

- Shift Taylor moments to the leaf nodes (L2L)



FMM Workflow

FMM Workflow

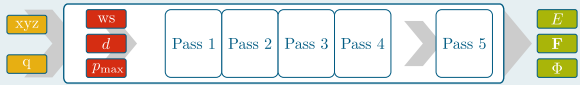


Pass 4

- Computation of the far field energy, forces, potentials

FMM Workflow

FMM Workflow



Pass 5

- Computation of the near field energy, forces, potentials

FMM Datastructures

Farfield

Coefficient Matrix, Generalized Storage Type

- Multipole order p
- Matrix size $\mathcal{O}(p^2)$
- Triangular shape

Multipole/Taylor Moments ω_{lm}, μ_{lm}

- Stored as coefficient matrix
- size $\mathcal{O}(p^2)$

Operators, e.g. M2L

- Stored as coefficient matrix
- size $\mathcal{O}(p^2)$

Translation Operation

$$\mu(\mathbf{b}-\mathbf{a}) = \sum_{l=0}^{p}\sum_{m=0}^{l}\sum_{j=0}^{p}\sum_{k=-j}^{j} \omega_{jk}(\mathbf{a}) O_{l+j,m+k}(b)$$

M2L Operation – FMM Tree Loops

Box – Neighbor Structure, ws=1

```
for (int d_c = 2; d_c <= depth; ++d_c) { // loop over all relevant tree levels

  for (int i = 0; i < dim; ++i) { // loop over all boxes on a certain level
    for (int j = 0; j < dim; ++j) {
      for (int k = 0; k < dim; ++k) {
         int idx = boxid(i, j, k);

         for (int ii = 0; ii < 6; ++ii) { // loop over all neighbors (216 for ws=1)
           for (int jj = 0; jj < 6; ++jj) {
             for (int kk = 0; kk < 6; ++kk) {

               int idx_n = neighbor_boxid(ii, jj, kk);
               int idx_op = operator_boxid(i, ii, j, jj, k, kk);

               // single p^4 operation
               M2L_operation(omega,*m2l_operators[d_c],mu, p, idx, idx_n, idx_op);
```

M2L Operation – Internal Structure

p^4 Loop structure

```
void M2L_operation(...){
  for (int l = 0; l <= p; ++l) {
    for (int m = 0; m <= l; ++m) {
      mu_l_m = 0;
      for (int j = 0; j <= p; ++j) {
        for (int k = -j; k <= j; ++k) {
          mu_l_m += M2L[idx_op](l + j, m + k) * omega[idx_n](j, k);
        }
      }
      mu[idx](l, m) += mu_l_m;
    }
  }
}
```

Parallelization Strategies

Full parallelization – all loops

Full parallel kernel

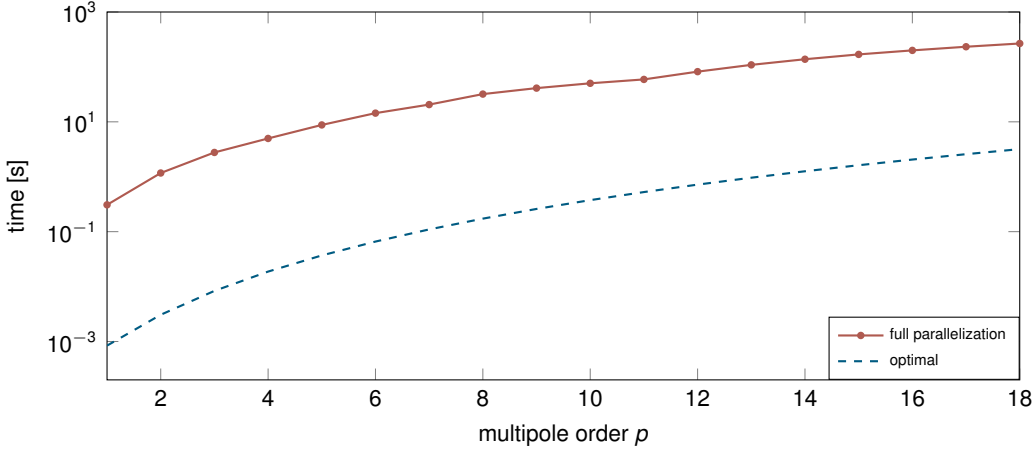
- Parallelize all loops
- Map last reduction loop to warps
- use `cub::WarpReduce` for reduction step `mu_l_m += ...`
- Drawbacks
  - Massive book keeping overhead
  - Execution time dominated by many integer divisions/modulo

Loop structure

```
for (int l = 0; l <= p; ++l)
  for (int m = 0; m <= l; ++m)
    mu_l_m = 0;
    for (int j = 0; j <= p; ++j)
      for (int k = -j; k <= j; ++k)
        mu_l_m += M2L[idx_op](l+j,m+k)
        *
        omega[idx_n](j, k);
    mu[idx](l, m) += mul_l_m
```

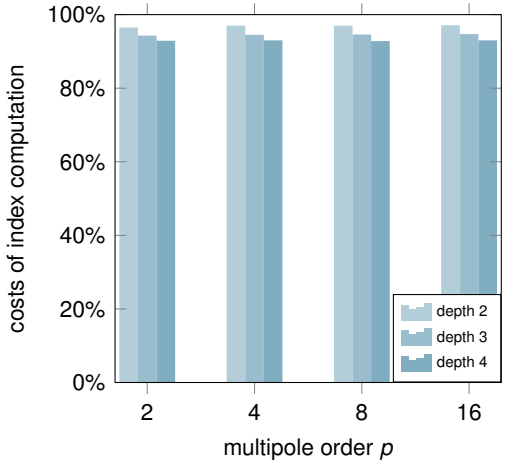
M2L Runtime – Full Loop Parallelization

Depth 4, 4096 Boxes



Full Parallelization Costs

Costs of index computing operations for M2L full parallel kernel

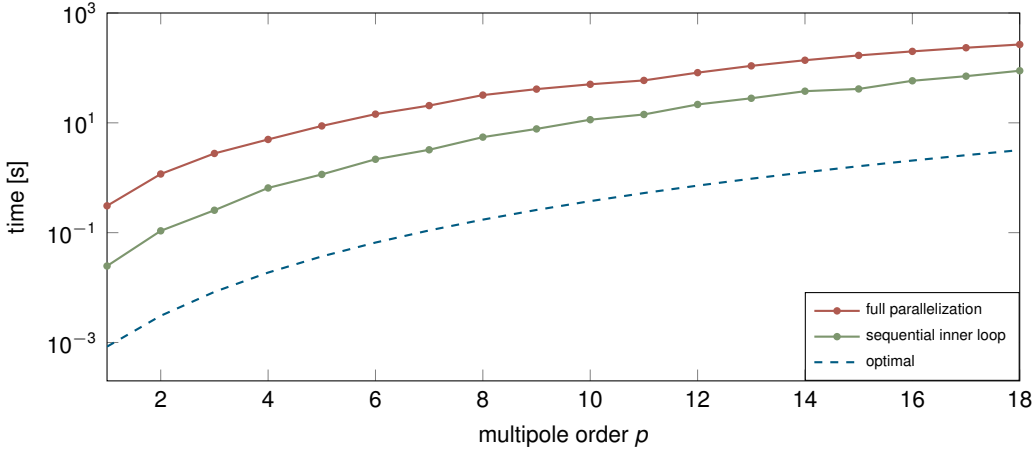


Reduce Overhead Further

- Keep innermost loop sequential
- Additional 2x – 7x speedup

M2L Runtime – Sequential Inner Loop

Depth 4, 4096 Boxes



Add Dynamic Parallelization I

Dynamic Scheme 1

Launch Kernels

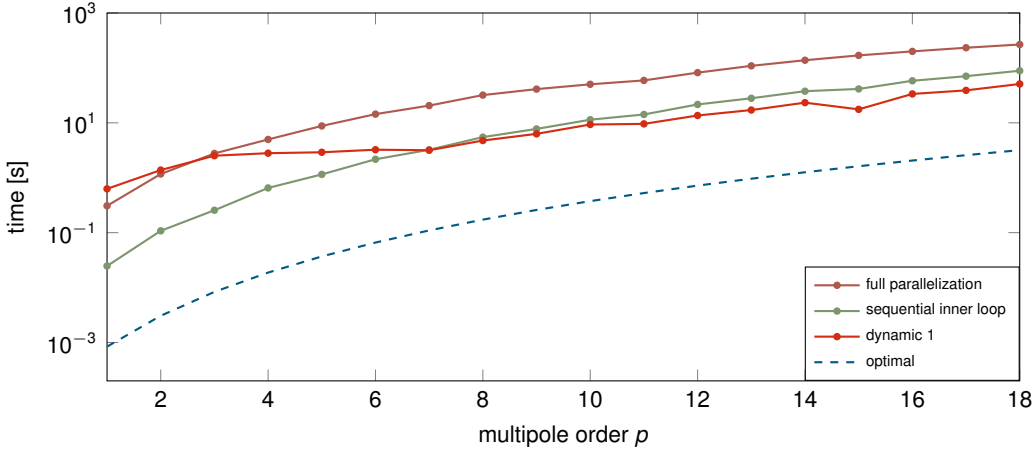
- loop over all tree level (2 – 10) @ host CPU
- loop over boxes on a certain level ($64, 512, 4096, \ldots$), launch kernel from host
- loop over all neighbor boxes (216), launch kernel from GPU (dynamically)
- launch kernel from GPU (dynamically) for each M2L operator

Drawbacks

- Large kernel launching latencies
- Kernels to small

M2L Runtime – Dynamic Parallelism I

Depth 4, 4096 Boxes



Add Dynamic Parallelization II

Dynamic Scheme 2

Launch Kernels

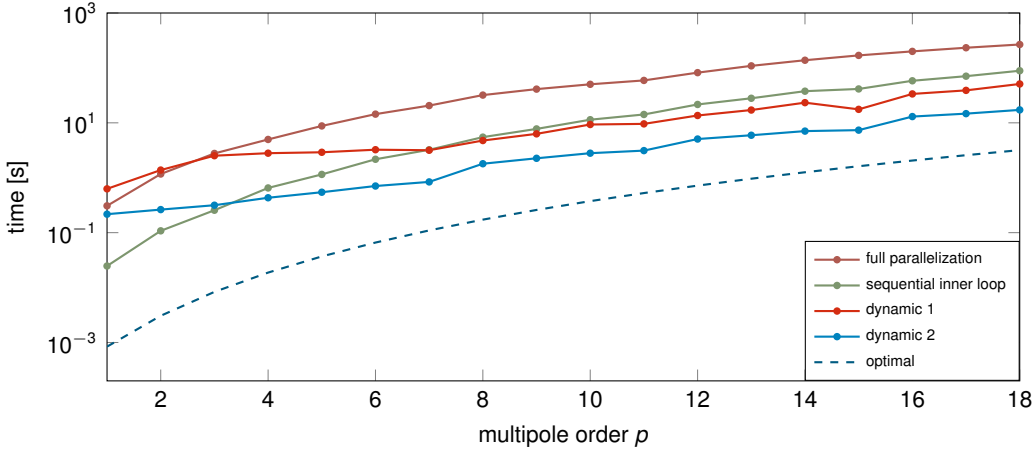
- loop over all tree level (2 – 10) @ host CPU
- loop over boxes on a certain level (64, 512, 4096, . . .) @ host CPU
- loop over all neighbor boxes (216), launch kernel from host
- launch kernel from GPU (dynamically) for each M2L operator

Drawbacks

- Global memory access too slow
- Disadvantageous access pattern

M2L Runtime – Dynamic Parallelism II

Depth 4, 4096 Boxes



Add Dynamic Parallelization III

Dynamic Scheme 3

Launch Kernels

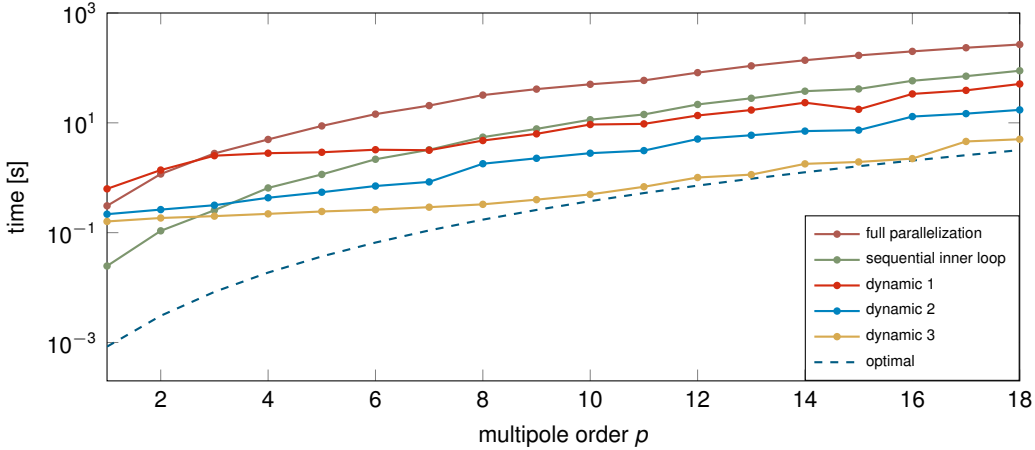
- loop over all tree level (2 – 10) @ host CPU
- loop over boxes on a certain level (64, 512, 4096, . . .) @ host CPU
- loop over all neighbor boxes (216), launch kernel from host
- launch kernel from GPU (dynamically) for each M2L operator

Improvements

- start only p^2 kernel threads
- Use shared GPU memory to cache ω and the operator M

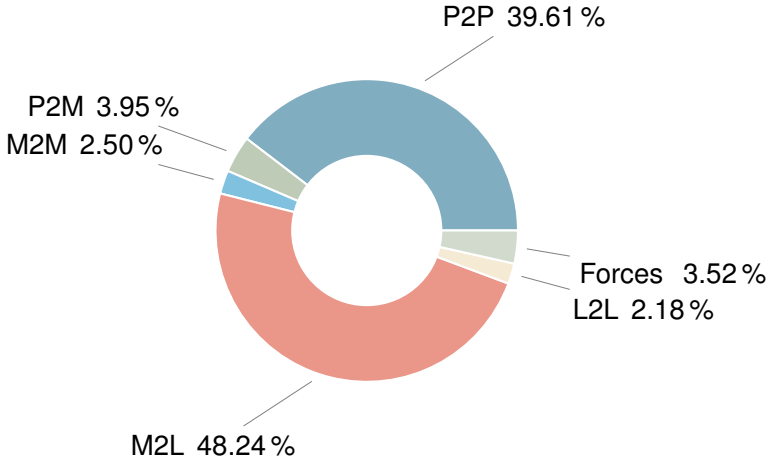
M2L Runtime – Dynamic Parallelism III

Depth 4, 4096 Boxes



Total Runtime Distribution

$N = 103000$, $p = 10$, $d = 4$, **Tesla K40**



Conclusions

The Smooth Part

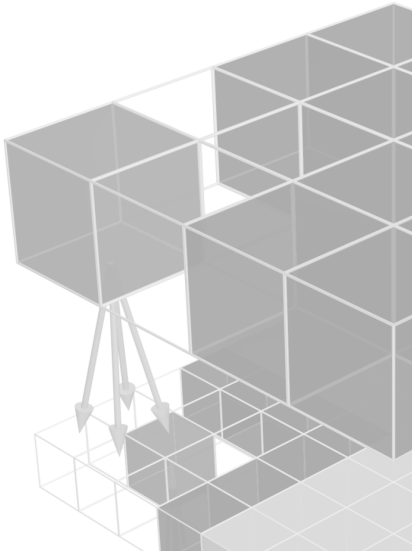
- Fairly fast start possible due to unified memory
- Dynamic parallelization avoids subsequent index computations (developer and hardware)
- Dynamic parallelization allows to achieve good performance (only for some predefined parameter setup)

The Bumpy Part

- Finding the best parallelization for every parameter setup is not trivial
- One 'fits all' kernel not possible (hybrid approach necessary)
- $\mathcal{O}(p^3)$ vs. $\mathcal{O}(p^4)$ operator GPU parallelization benefit?

Questions?

bartosz.kohnke@mpibpc.mpg.de



FMM goes GPU

A smooth trip or bumpy ride?

March 18, 2015 | B. Kohnke, I.Kabadshow | MPI BPC Göttingen & Jülich Supercomputing Centre