IMA Annual Program Year Workshop:

High Performance Computing and Emerging Architectures

Minneapolis, January 10-14, 2011

The basis and perspectives of an exascale algorithm: our ExaFMM project.

Lorena A Barba, Boston University

BU College of Engineering

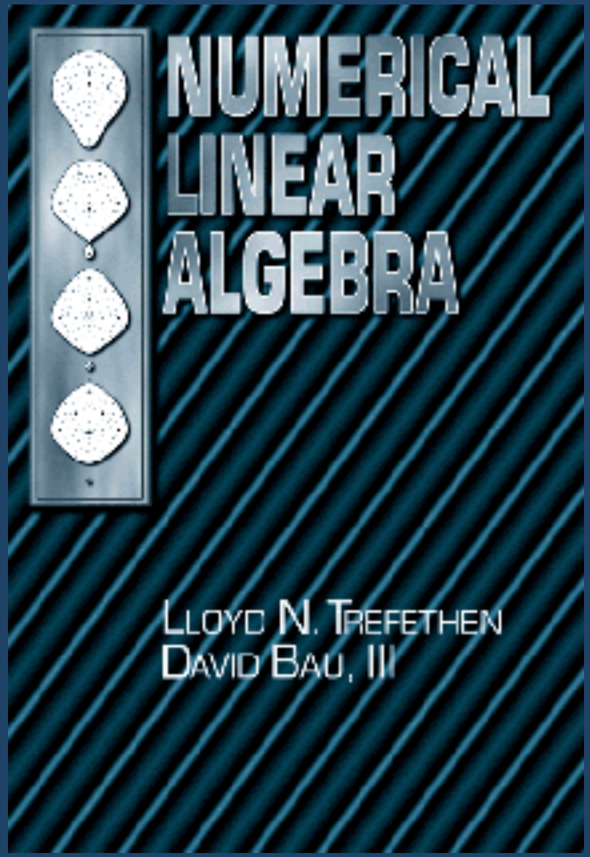
Acknowledgements:

work in Barba's group done in collaboration with
Jaydeep Bardhan (Rush), Mathew Knepley (UChicago),
Tsuyoshi Hamada (Nagasaki Advanced Computing Center),
Rio Yokota (postdoc at BU) and graduate students
Felipe Cruz, Christopher Cooper, Anush Krishnan, Simon Layton

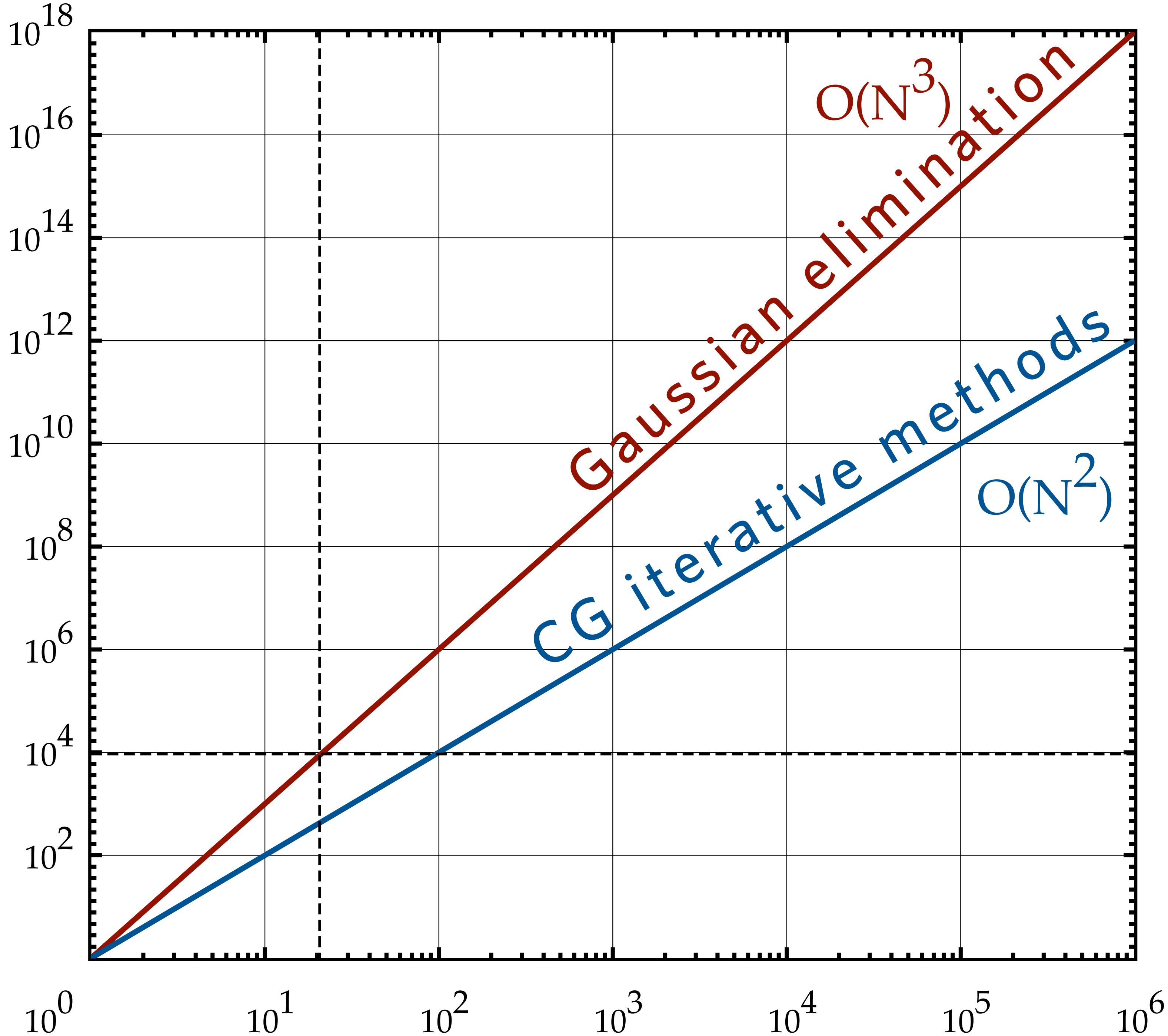
in Nagasaki Advanced Computing Center

“… the fundamental law of computer science [is]: the faster the computer, the greater the importance of speed of algorithms”

Trefethen & Bau “Numerical Linear Algebra” SIAM

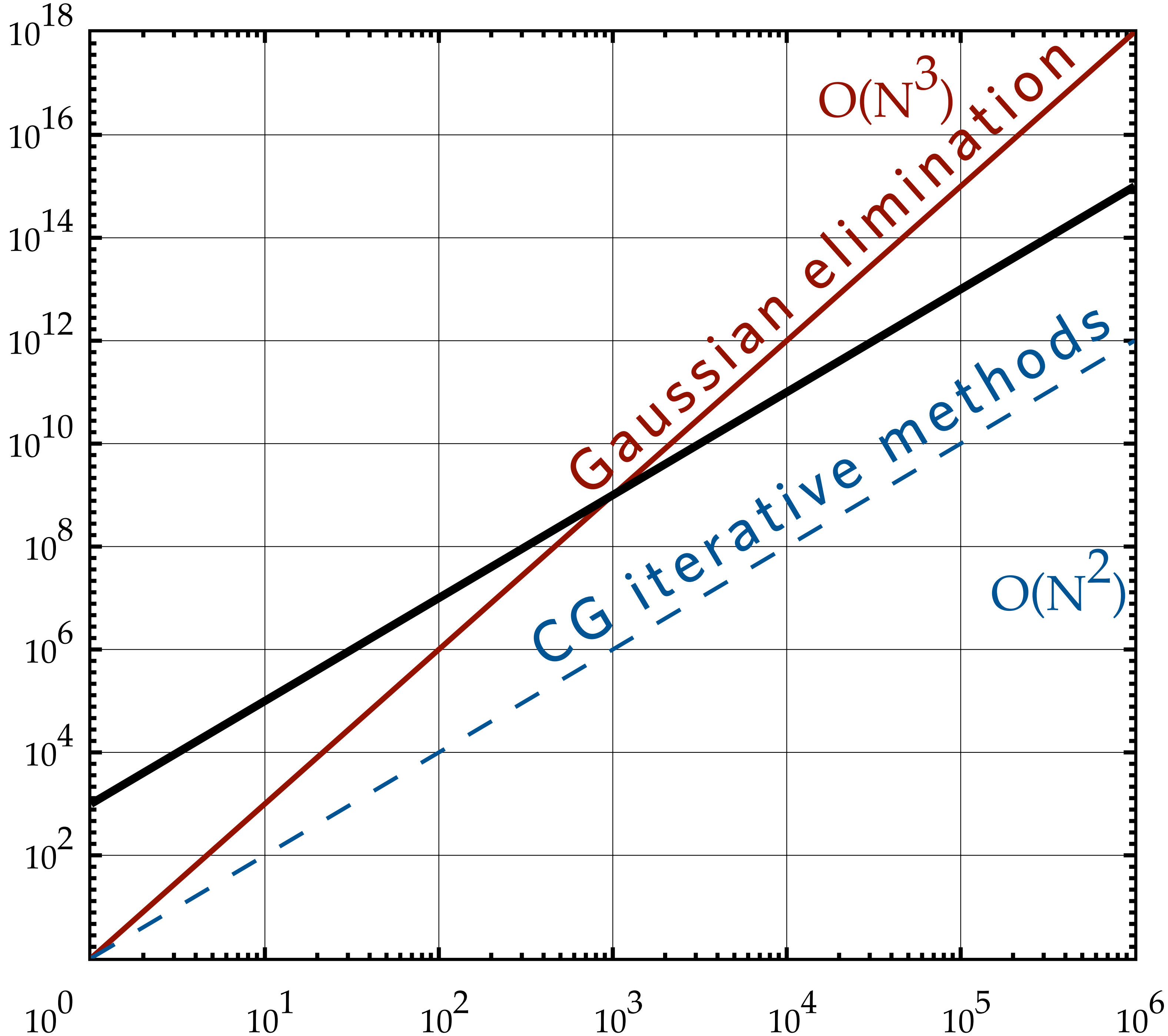


The curious story of conjugate gradient (CG) algorithms



- Iterative methods:
  - *sequence of iterates converging to the solution*
- CG matrix iterations bring the $O(N^3)$ cost to $O(N^2)$
- 1950s — ***N too small*** for CG to be competitive
- 1970s — renewed attention

The curious story of conjugate gradient (CG) algorithms



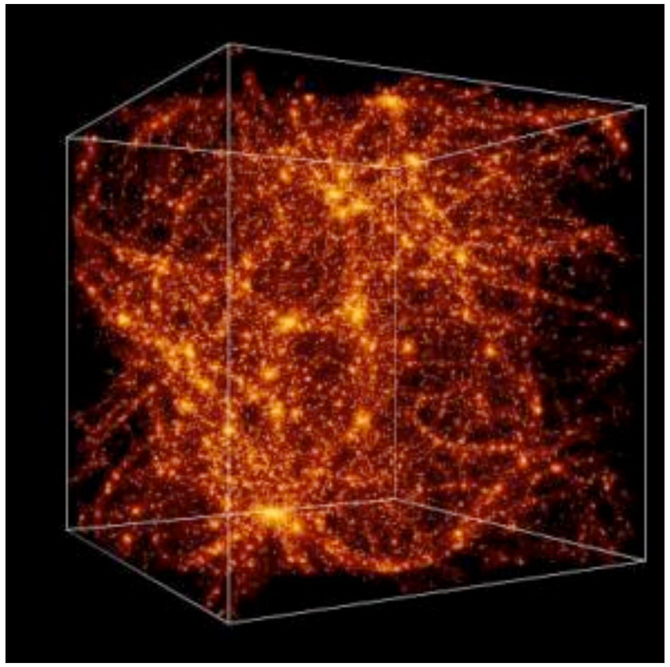
- Iterative methods:
  - *sequence of iterates converging to the solution*
- CG matrix iterations bring the $O(N^3)$ cost to $O(N^2)$
- 1950s — ***N too small*** for CG to be competitive
- 1970s — renewed attention

- 1946 — The Monte Carlo method.
- 1947 — Simplex Method for Linear Programming.
- 1950 — **Krylov Subspace Iteration Method**.
- 1951 — The Decompositional Approach to Matrix Computations.
- 1957 — The Fortran Compiler.
- 1959 — QR Algorithm for Computing Eigenvalues.
- 1962 — Quicksort Algorithms for Sorting.
- 1965 — Fast Fourier Transform.
- 1977 — Integer Relation Detection.
- 1987 — **Fast Multipole Method**

Dongarra& Sullivan, IEEE Comput. Sci. Eng., Vol. 2(1):22-- 23 (2000)

Fast multipole method

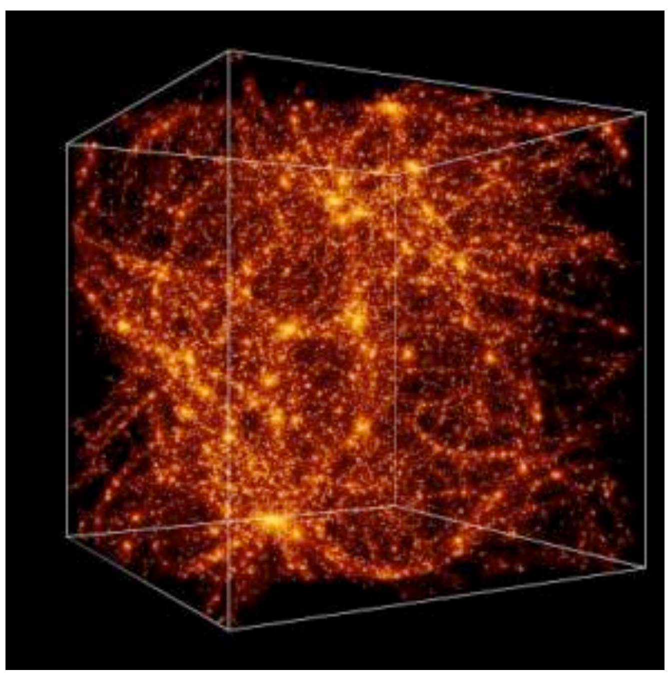
- Solves N-body problems
  - e.g. astrophysical gravity interactions
  - reduces operation count from **$O(N^2)$ to $O(N)$**



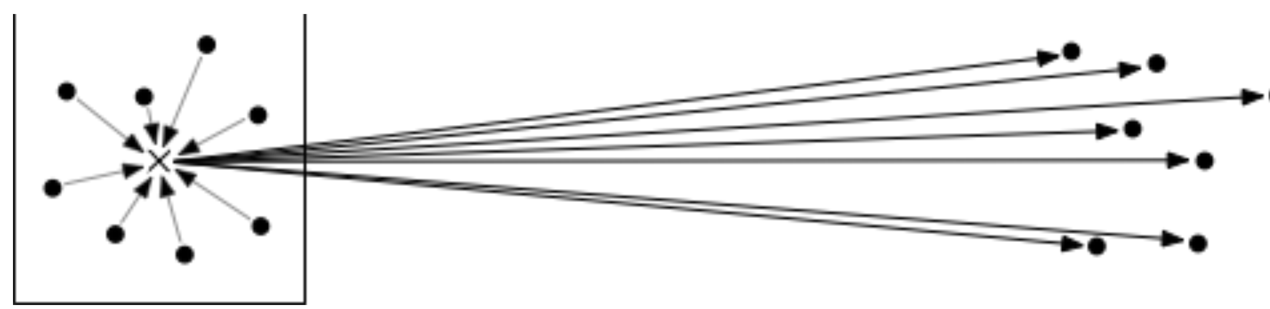
$$f(y) = \sum_{i=1}^{N} c_i \mathbf{K}(y - x_i) \qquad y \in [1...N]$$

Fast multipole method

- Solves N-body problems
  - e.g. astrophysical gravity interactions
  - reduces operation count from **O(N²) to O(N)**

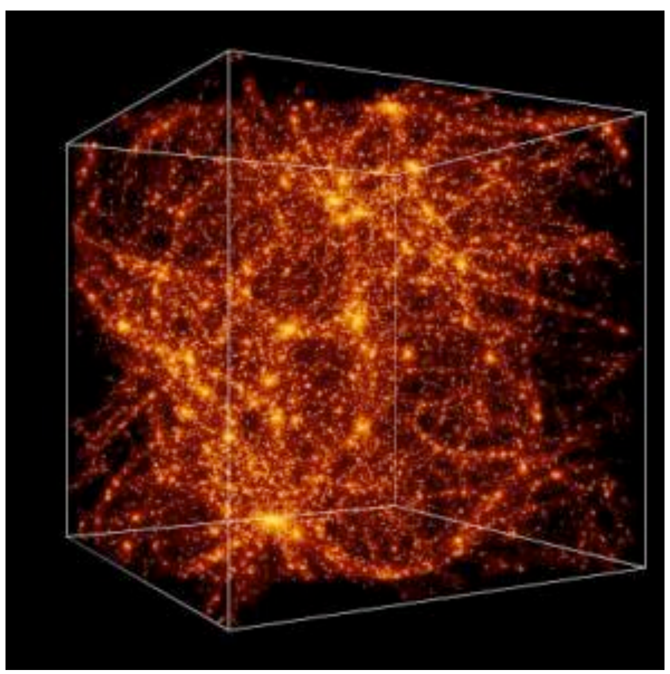


$$f(y) = \sum_{i=1}^{N} c_i \mathbf{K}(y - x_i) \qquad y \in [1...N]$$



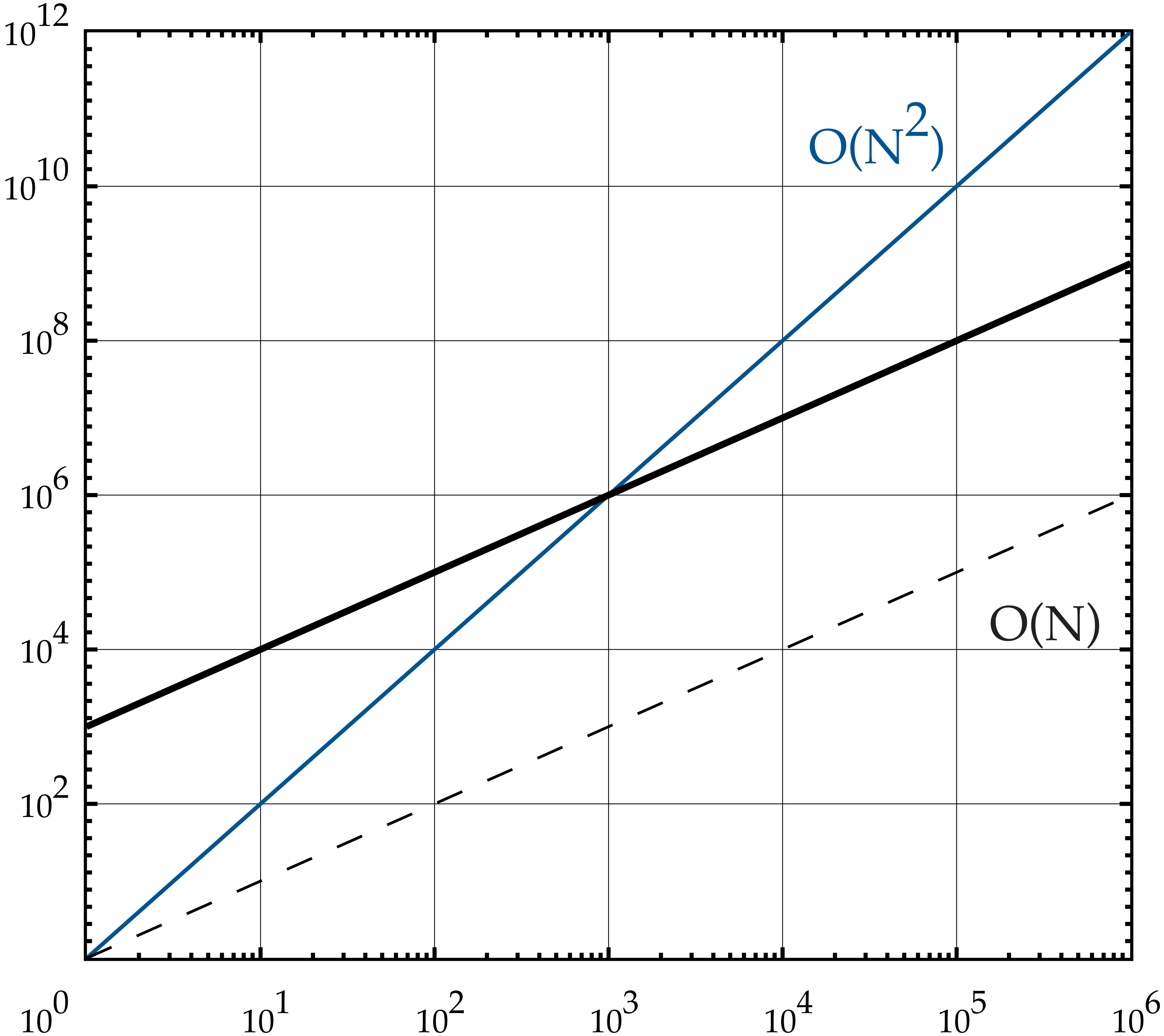
Fast multipole method

- Solves N-body problems
  - e.g. astrophysical gravity interactions
  - reduces operation count from **$O(N^2)$ to $O(N)$**



$$f(y) = \sum_{i=1}^{N} c_i \mathbf{K}(y - x_i) \qquad y \in [1 \dots N]$$

O(N) advantage



Hierarchical methods:

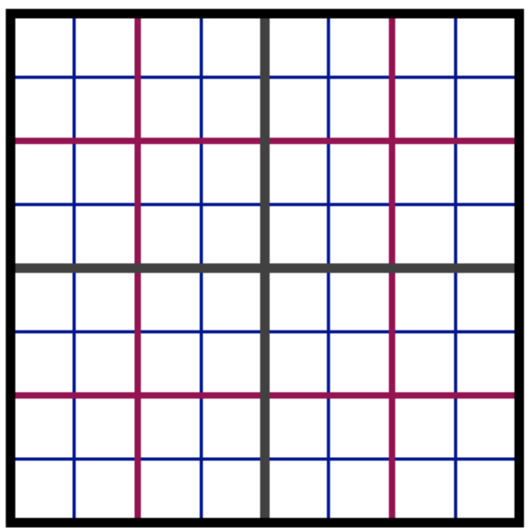
- *sequence of refinements converging (or contributing) to the solution*

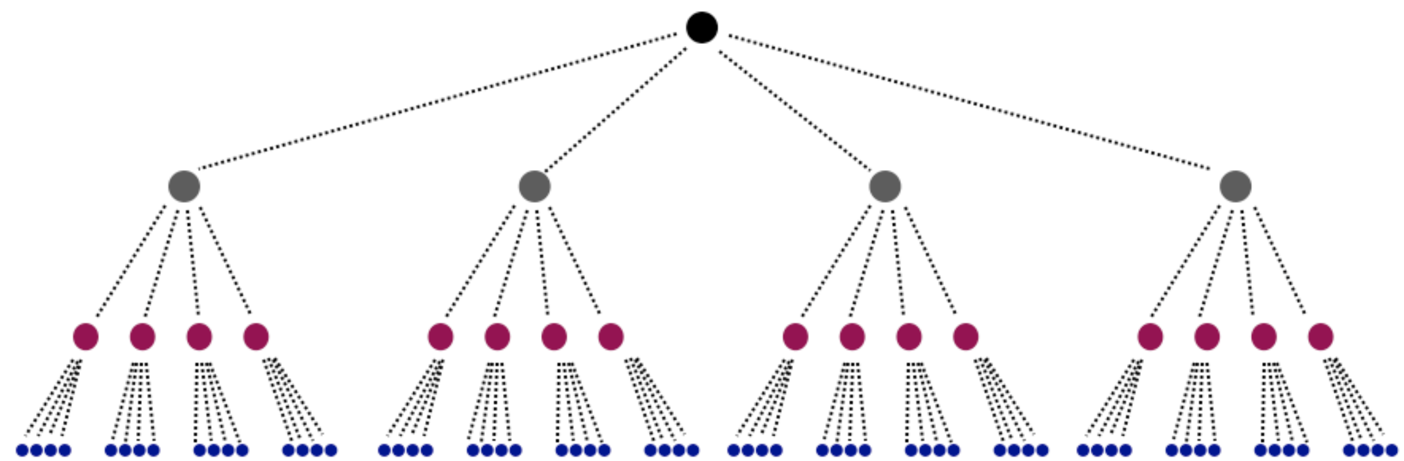
FMM brings the $O(N^2)$ cost to O(N)

1990s — MD codes dropped FMM, as ***N too small*** to be competitive

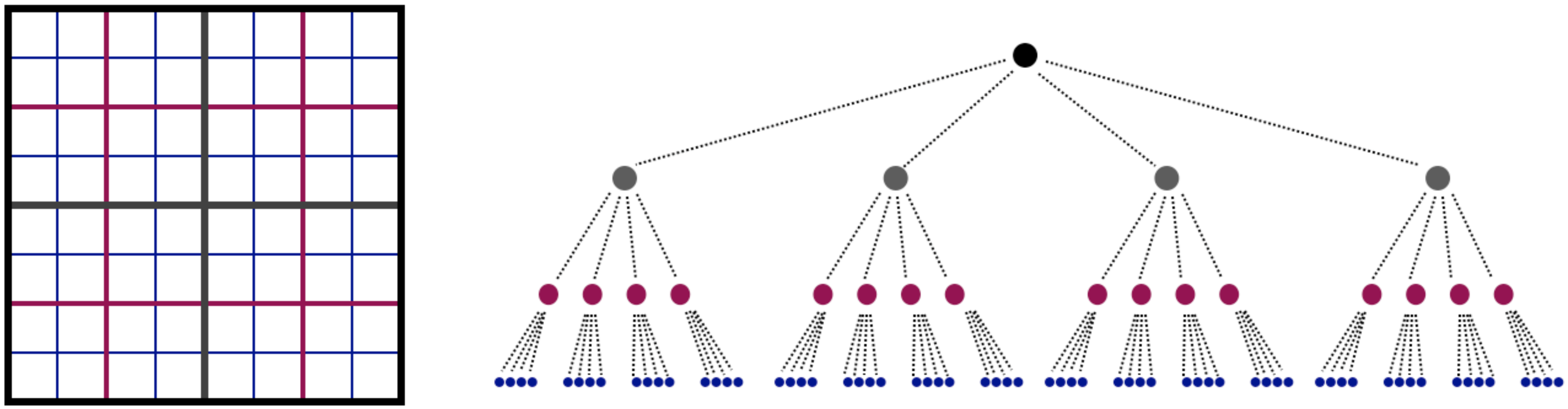
Now — renewed attention

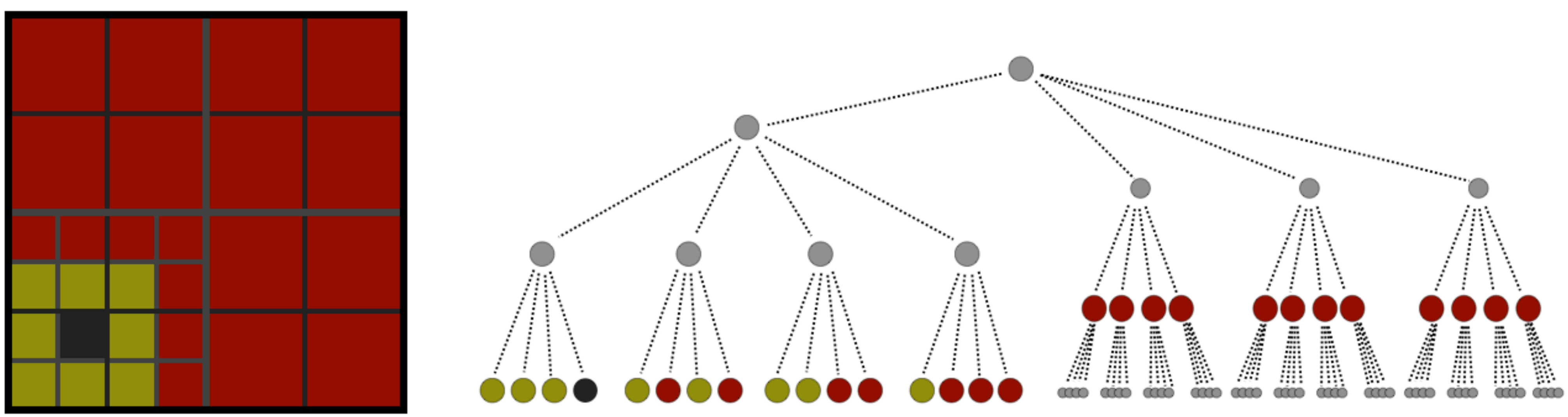
- space subdivision tree structure
  - to find “near” and “far” bodies



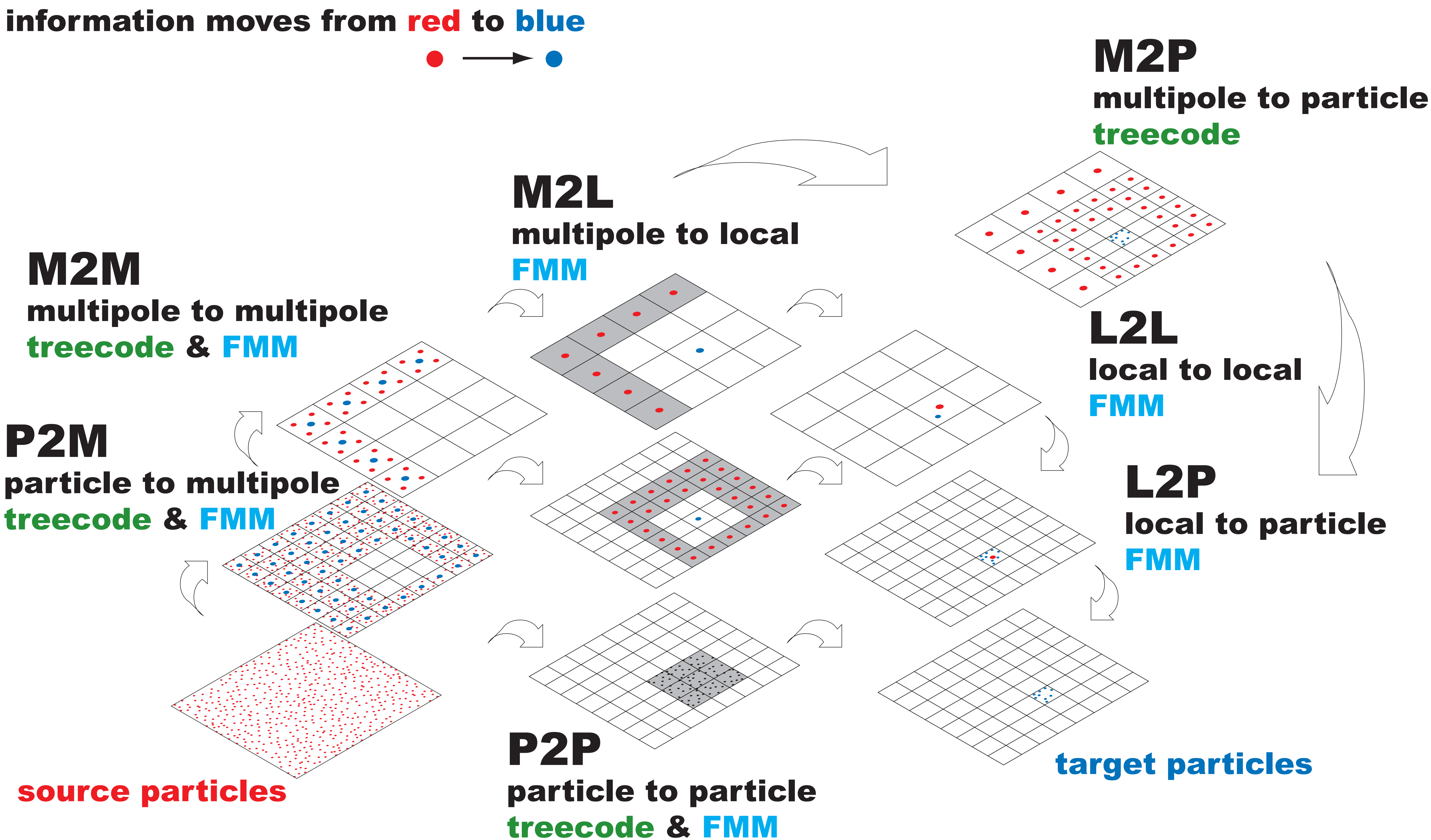


- space subdivision tree structure
  - to find “near” and “far” bodies

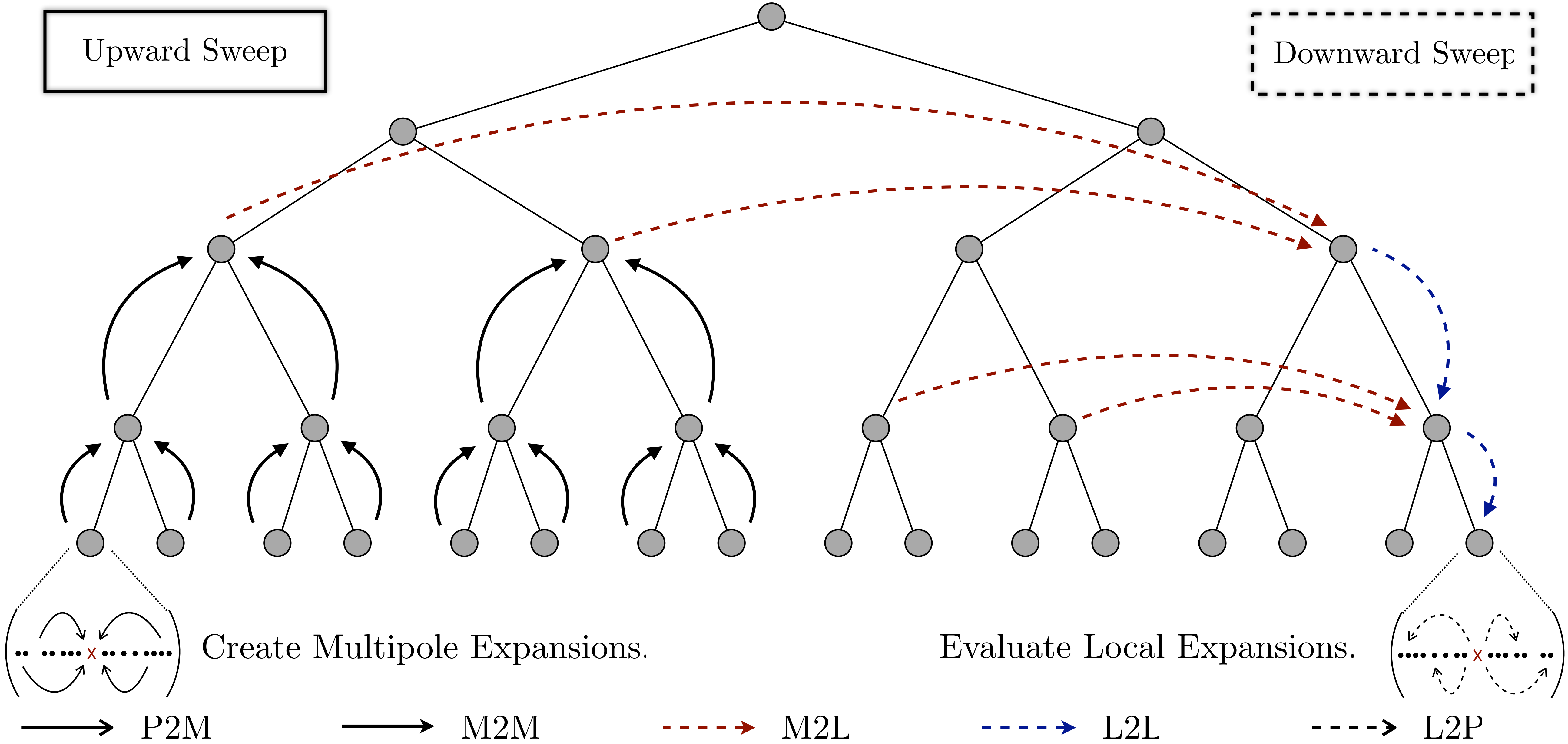




Flow of FMM calculation



▸ The whole algorithm in a sketch



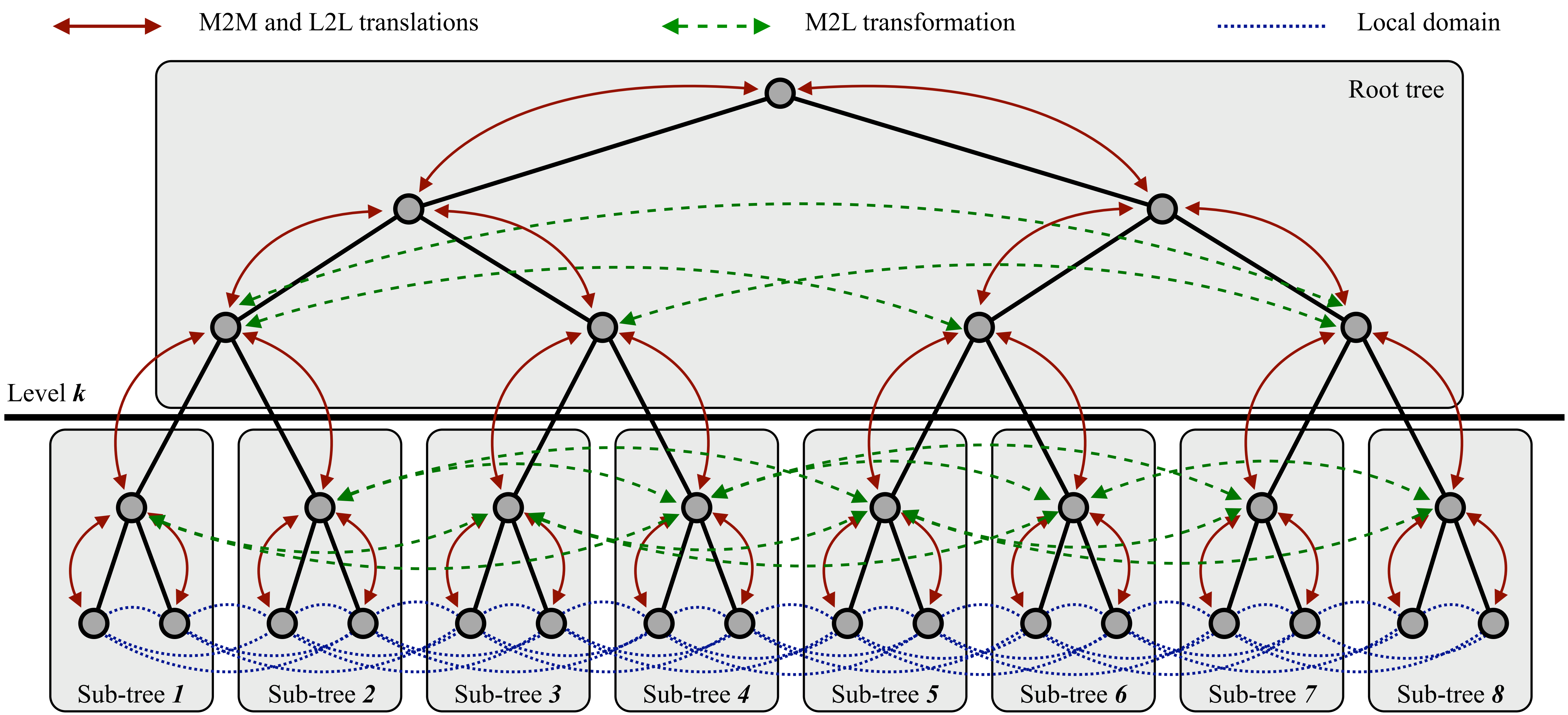
- *Contributions from Barba group:*

INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2011; **85**:403–428
Published online 1 September 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nme.2972

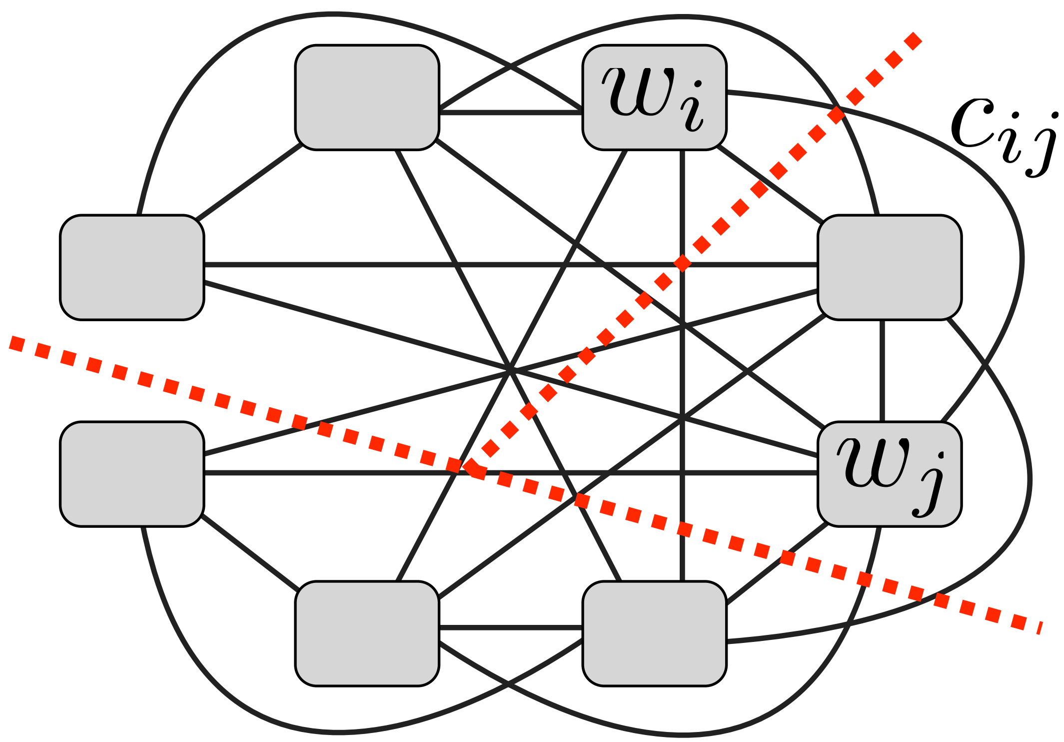
PetFMM—A dynamically load-balancing parallel fast multipole library

Felipe A. Cruz[1], Matthew G. Knepley[2] and L. A. Barba[3,*,†]

- Parallelization strategy:



- Graph representation:



Ref. — F. A Cruz, M. G. Knepley, L. A. Barba,
PetFMM—A dynamically load-balancing parallel fast multipole library,
Int. J. Num. Meth. Eng., Vol. 85(4): 403–428 (Jan. 2011)

GPU implementation of FMM kernels

The algorithmic and hardware speed-ups properly multiply

Treecode and fast multipole method for N-body simulation with CUDA

Rio Yokota
Boston University

Lorena A. Barba
Boston University

GPU Gems, Volume IV

In press, to appear February 2011 (?)

Codes in http://code.google.com/p/gemsfmm/

Chapter 31

Fast N-Body Simulation with CUDA

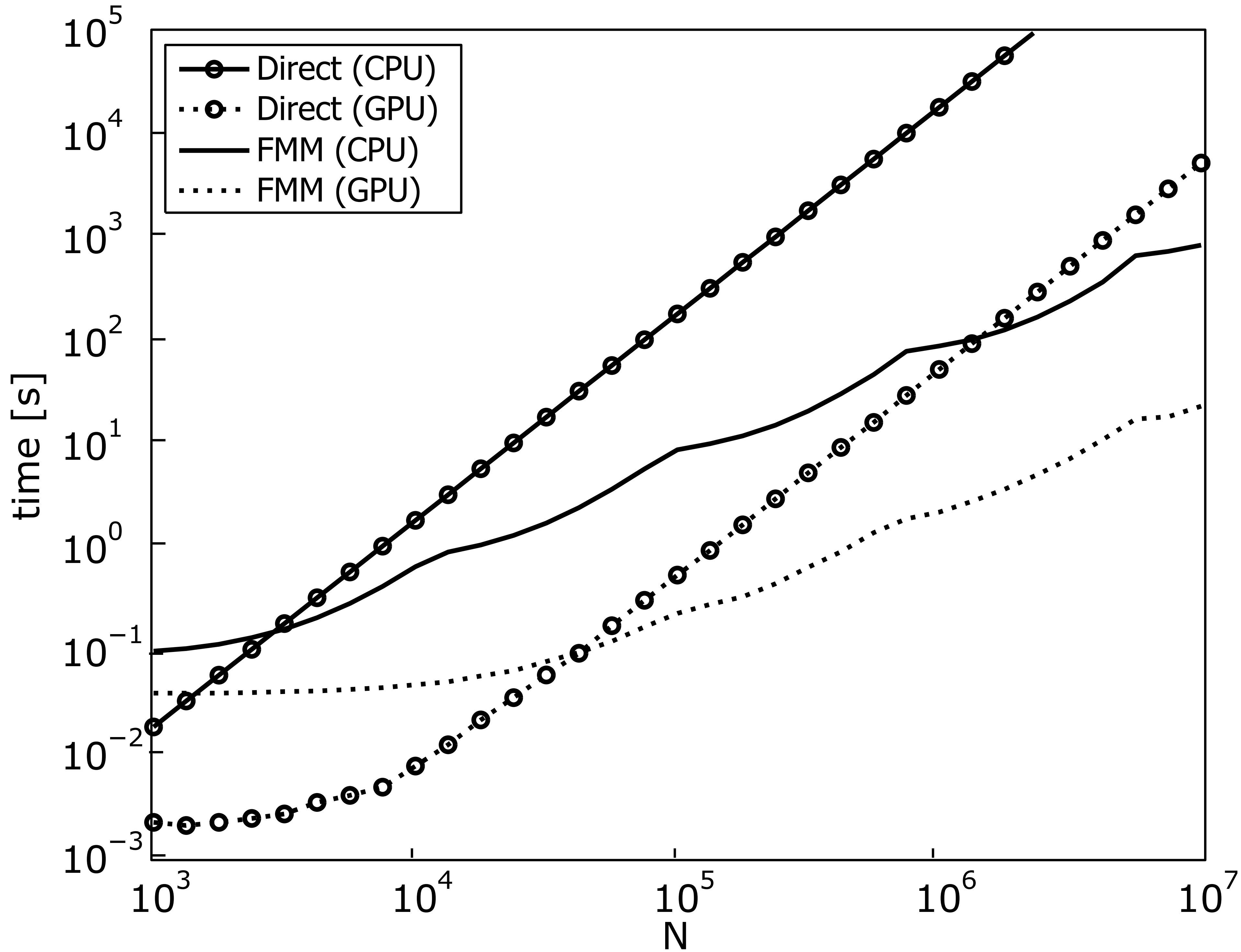
Lars Nyland
NVIDIA Corporation

Mark Harris
NVIDIA Corporation

GPU Gems, Volume III

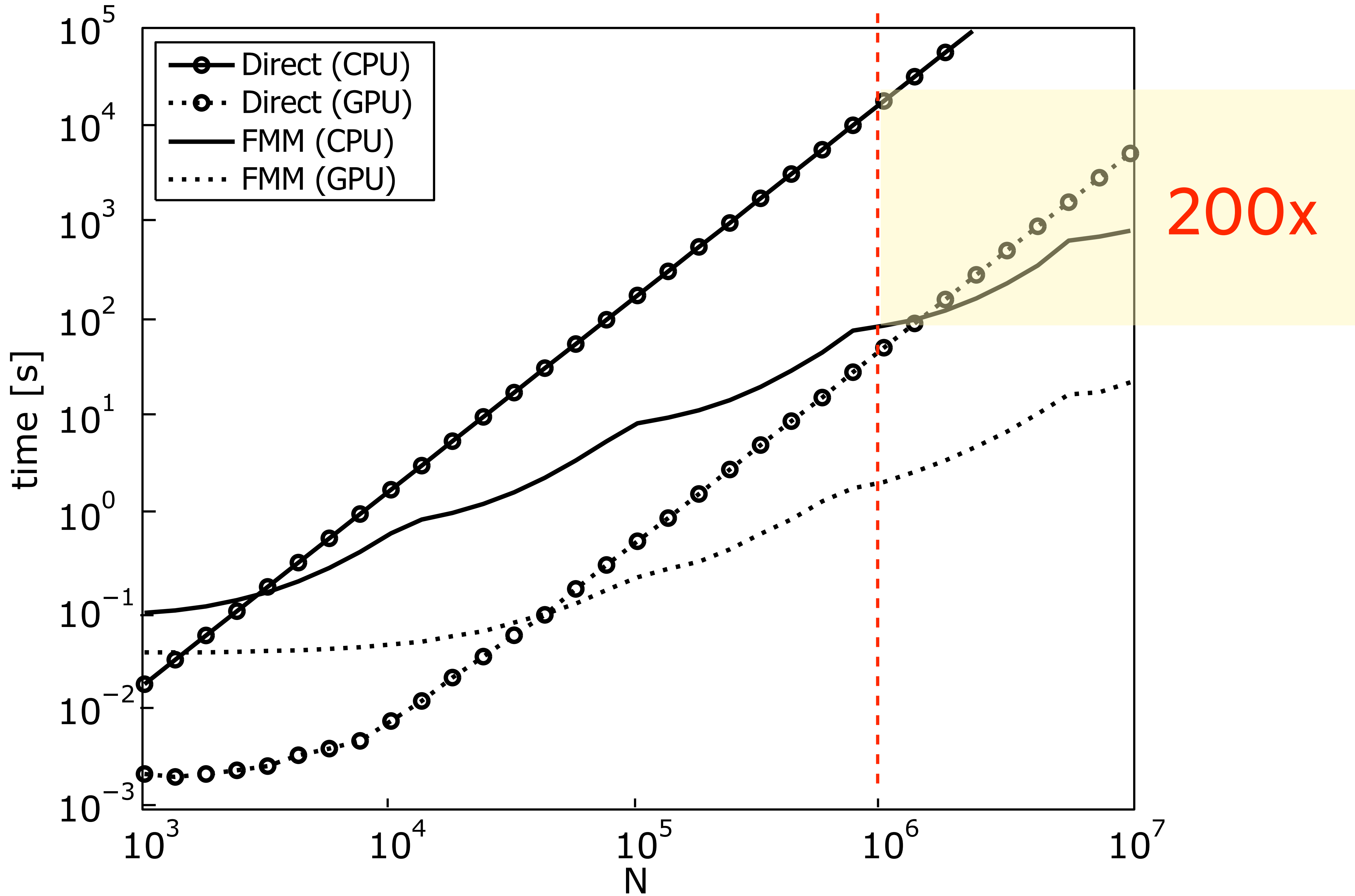
FMM on GPU

"Treecode and fast multipole method for N-body simulation with CUDA", chapter in ***GPU Gems IV***, in press



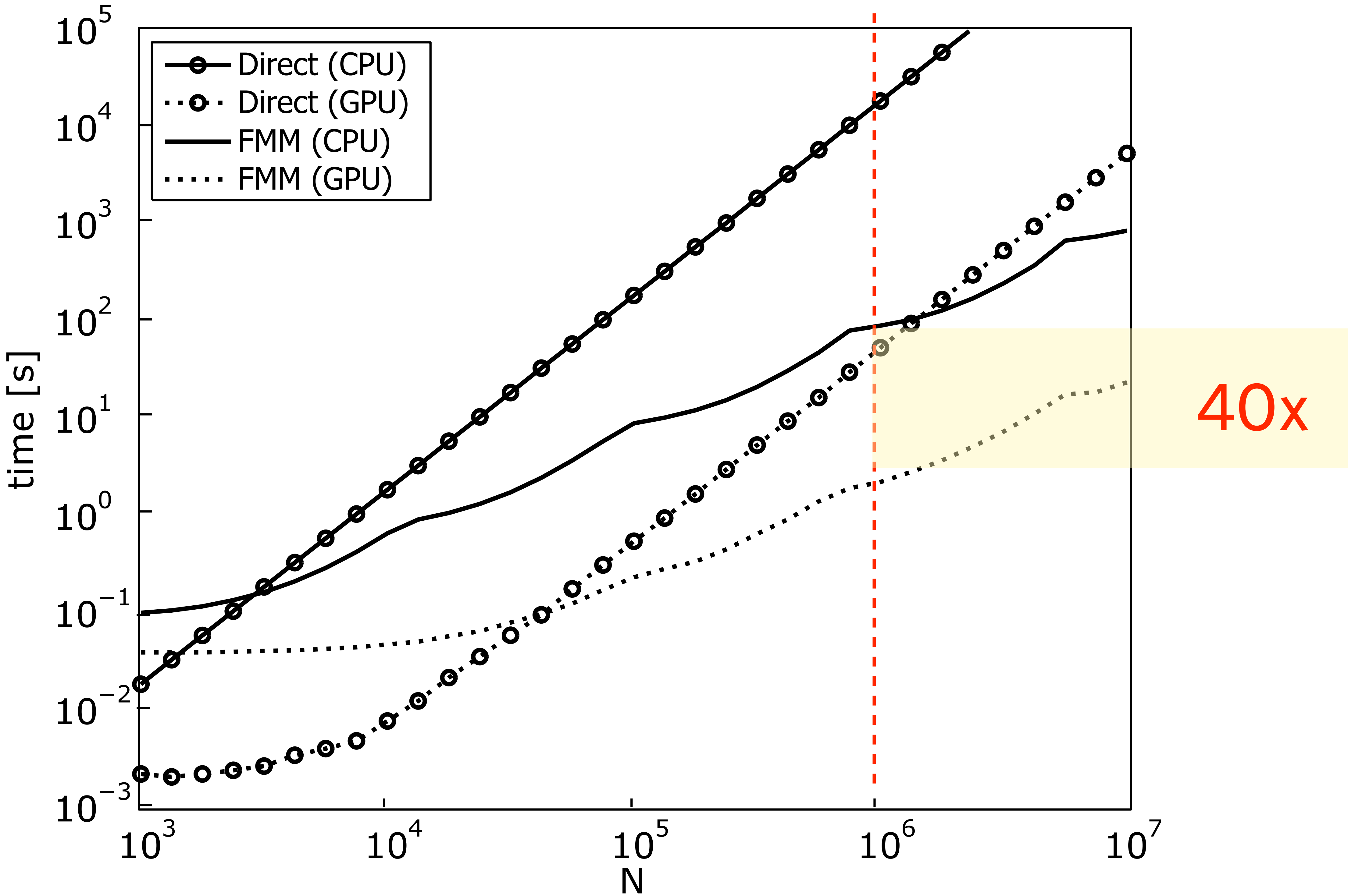
FMM on GPU

"Treecode and fast multipole method for N-body simulation with CUDA", chapter in ***GPU Gems IV***, in press



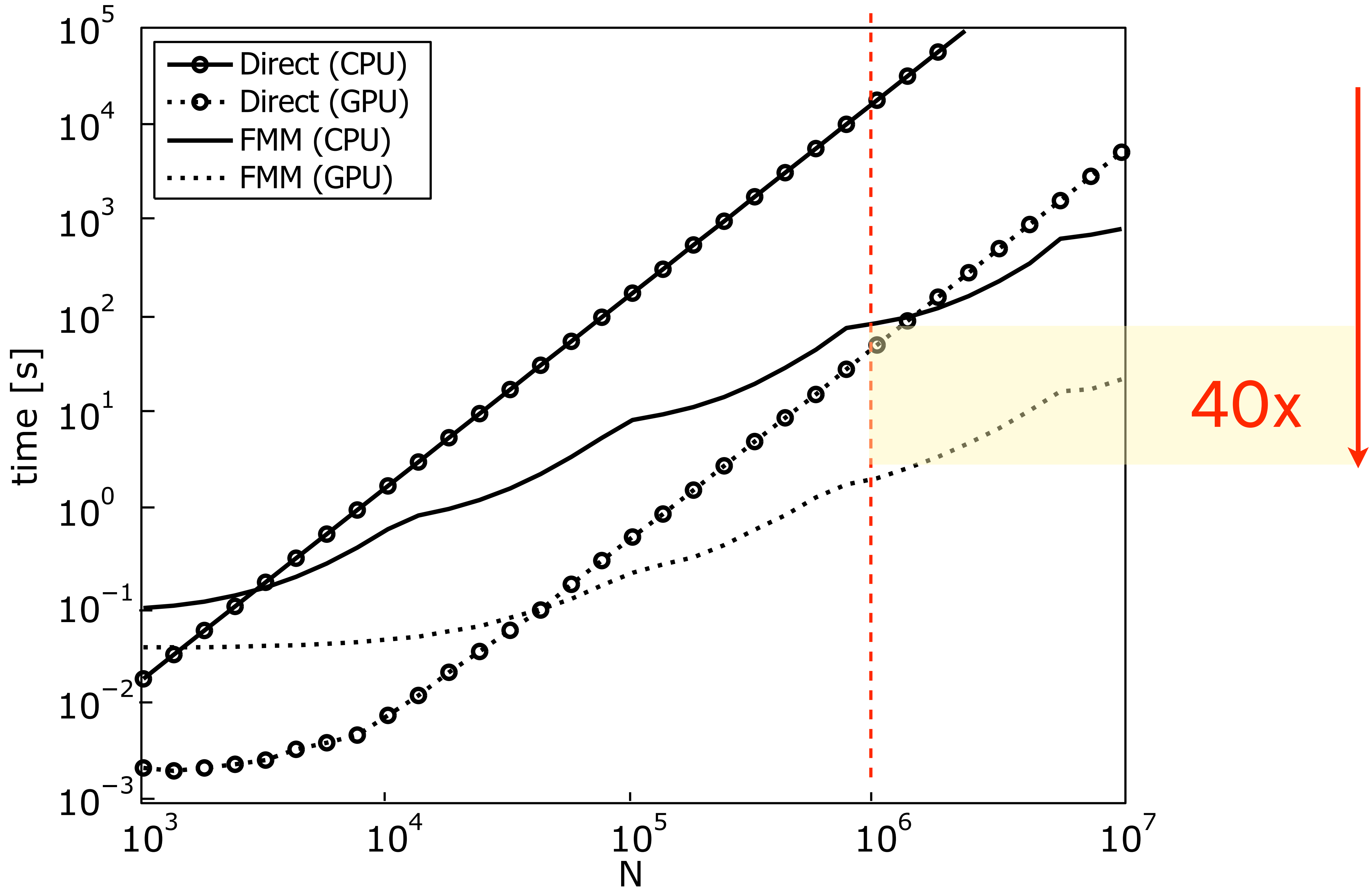
FMM on GPU

"Treecode and fast multipole method for N-body simulation with CUDA", chapter in ***GPU Gems IV***, in press



FMM on GPU

"Treecode and fast multipole method for N-body simulation with CUDA", chapter in *GPU Gems IV*, in press



- *the right* ***methods and algorithms*** *can provide leaps in capability many times that of Moore's law would in a given period*

- ***open source & open data*** *enables tackling large, complex computational projects*

- *the right* ***methods and algorithms*** *can provide leaps in capability many times that of Moore's law would in a given period*
- ***open source & open data*** *enables tackling large, complex computational projects*
- ***new hardware*** *for HPC adds to the mix for a new era of discovery via computation*

Parallel FMM on multi-GPUs

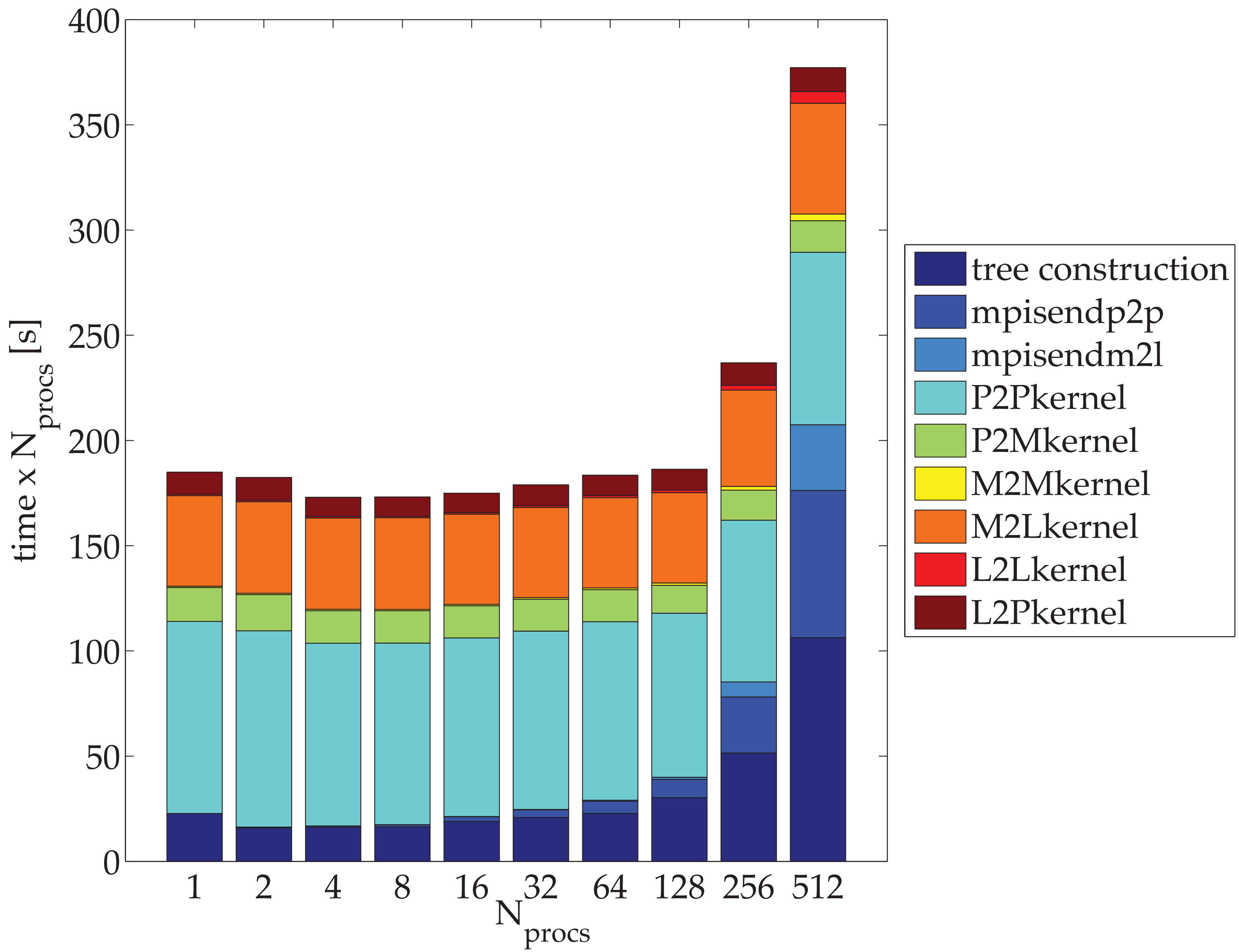
Strong Scaling:

parallel efficiency of 80% at 256, and 50% at 512 nodes

$N=10^8$

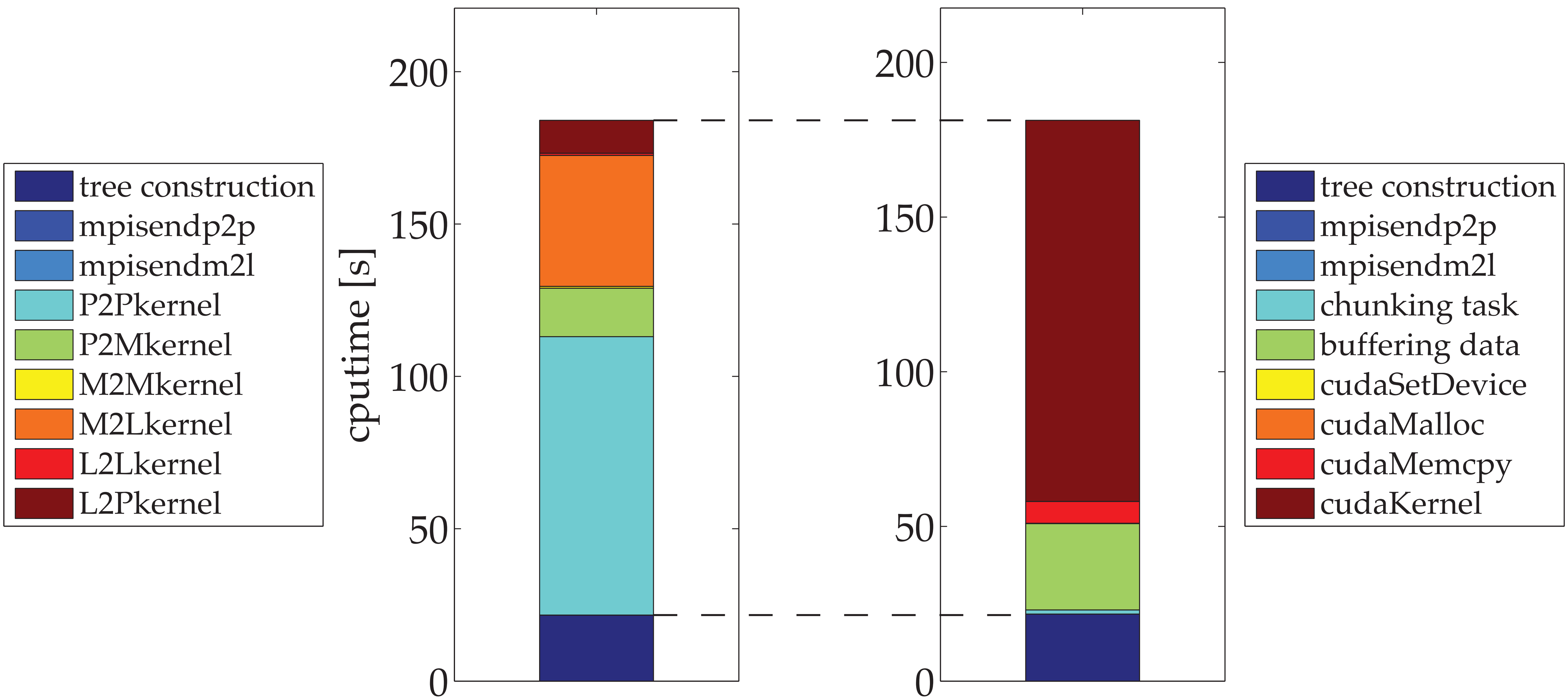
p=10

Degima cluster at NACC, with Infiniband comm



GPU breakdown

- $N=10^8$, on one node



Cornell University Library

arXiv.org > cs > arXiv:1007.4591

Computer Science > Computational Engineering, Finance, and Science

Biomolecular electrostatics simulation with a parallel FMM-based BEM, using up to 512 GPUs

Rio Yokota, Jaydeep P. Bardhan, Matthew G. Knepley, L. A. Barba, Tsuyoshi Hamada

(Submitted on 26 Jul 2010 (v1), last revised 17 Oct 2010 (this version, v2))

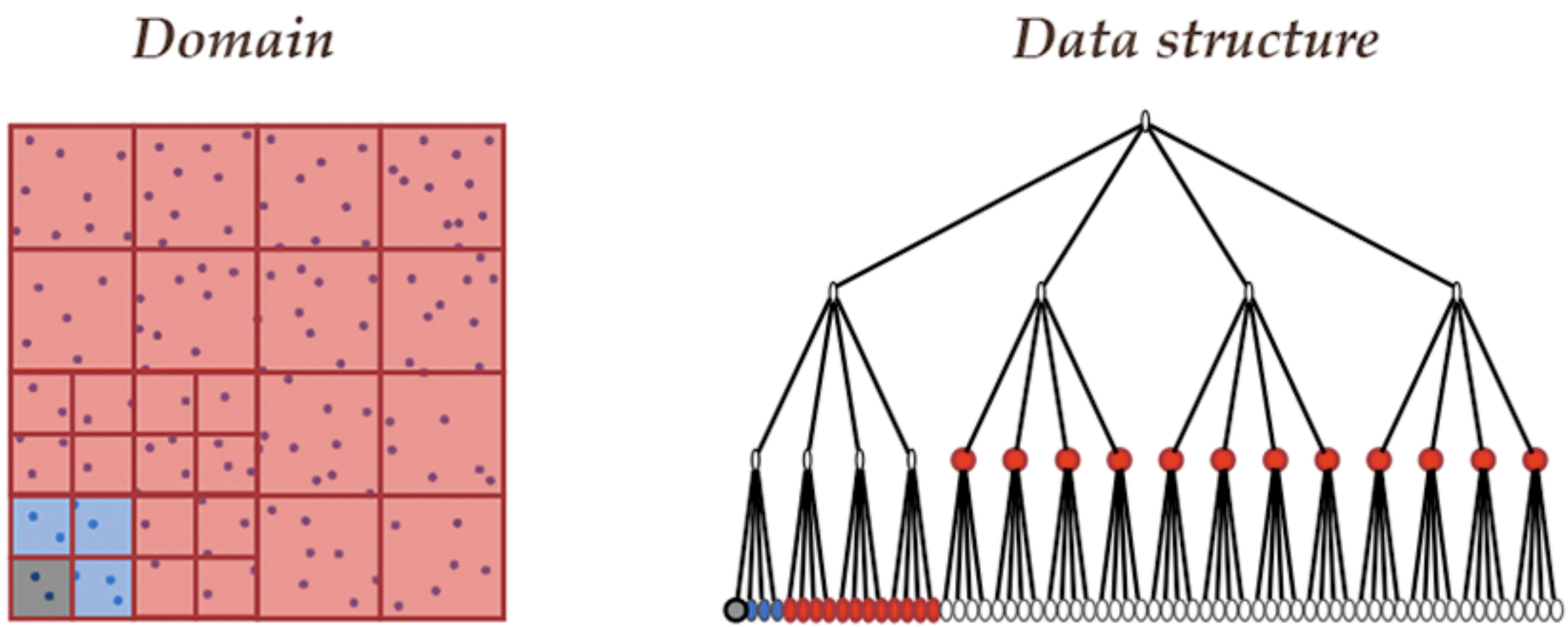
Under revision for *Comput. Phys. Comm.*

See also `http://barbagroup.bu.edu/`

Suitability of the FMM for achieving exascale

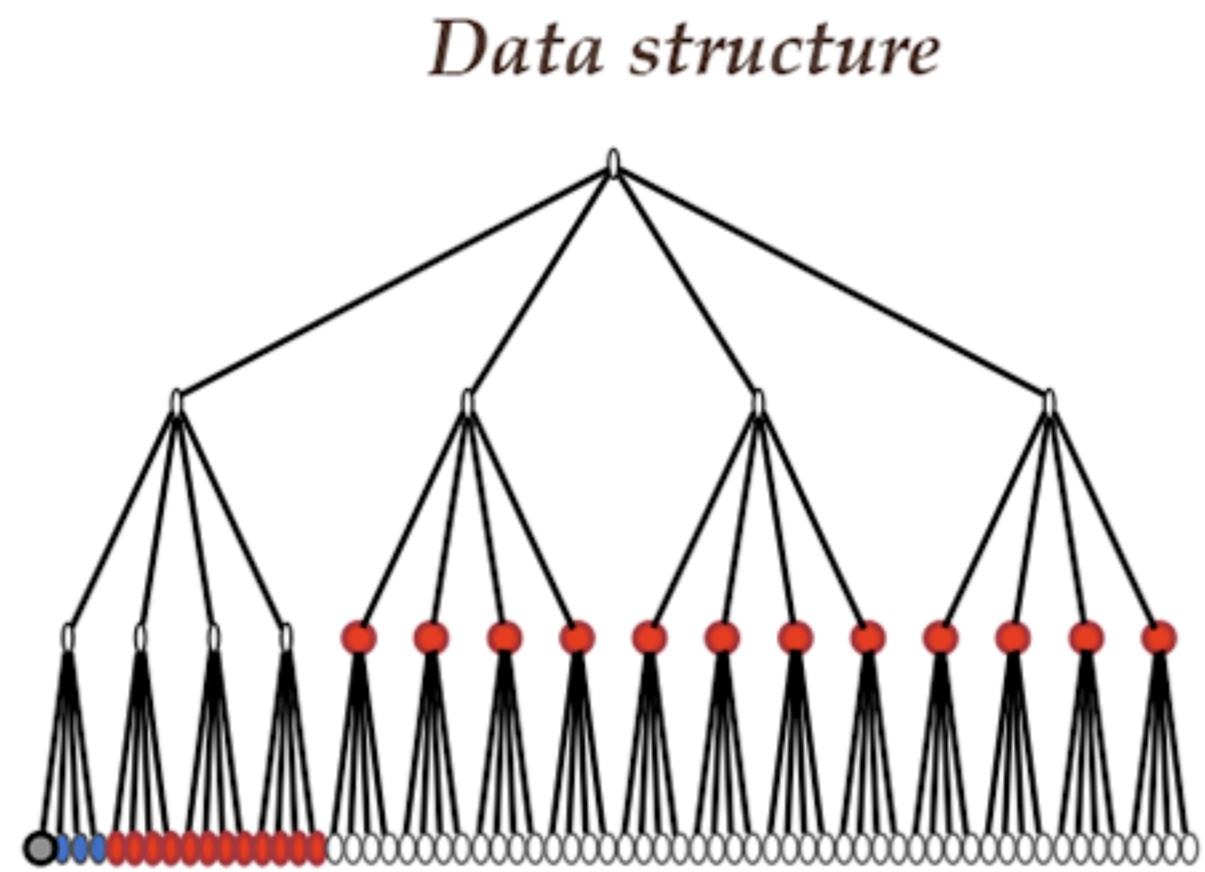
FMM is a particularly favorable algorithm for the emerging heterogeneous, many-core architectural landscape.

Spatial and temporal locality



Spatial and temporal locality

- Algorithm has intrinsic geometric locality



Spatial and temporal locality

- Algorithm has intrinsic geometric locality

Spatial and temporal locality

- Algorithm has intrinsic geometric locality
- Acces patterns *could* be non-local

Spatial and temporal locality

- Algorithm has intrinsic geometric locality
- Acces patterns *could* be non-local
  - work with sorted particle indices, access via a start-offset combination

Spatial and temporal locality

- Algorithm has intrinsic geometric locality
- Acces patterns *could* be non-local
  - work with sorted particle indices, access via a start-offset combination
- Temporal locality:

Spatial and temporal locality

- Algorithm has intrinsic geometric locality
- Acces patterns *could* be non-local
    - work with sorted particle indices, access via a start-offset combination
- Temporal locality:
    - queue GPU tasks before execution, buffer the input and output of data making memory access contiguous

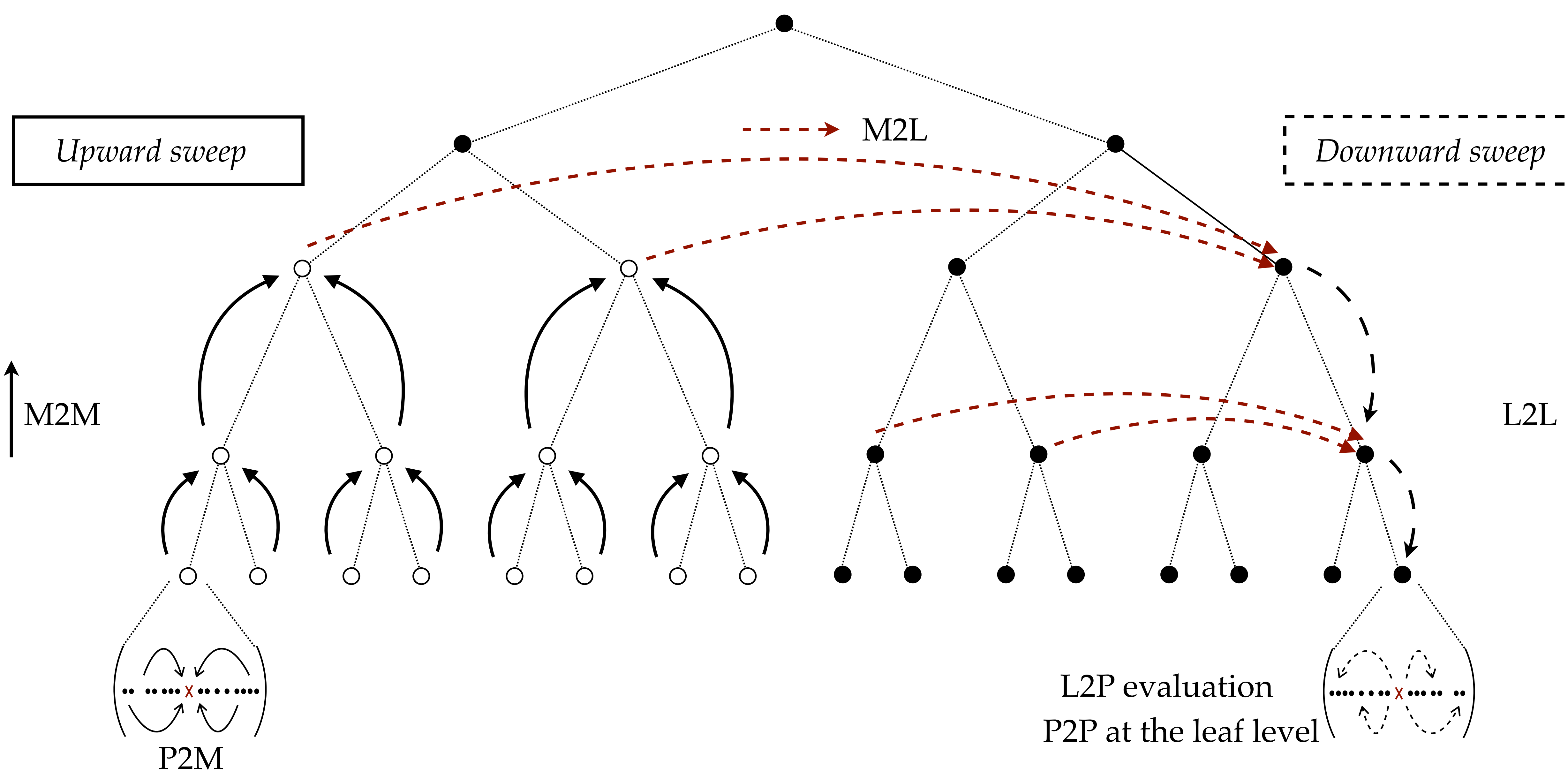
Spatial and temporal locality

- Algorithm has intrinsic geometric locality
- Acces patterns *could* be non-local
  - work with sorted particle indices, access via a start-offset combination
- Temporal locality:
  - queue GPU tasks before execution, buffer the input and output of data making memory access contiguous

➡ *The FMM is **not** a locallity-sensitive application*

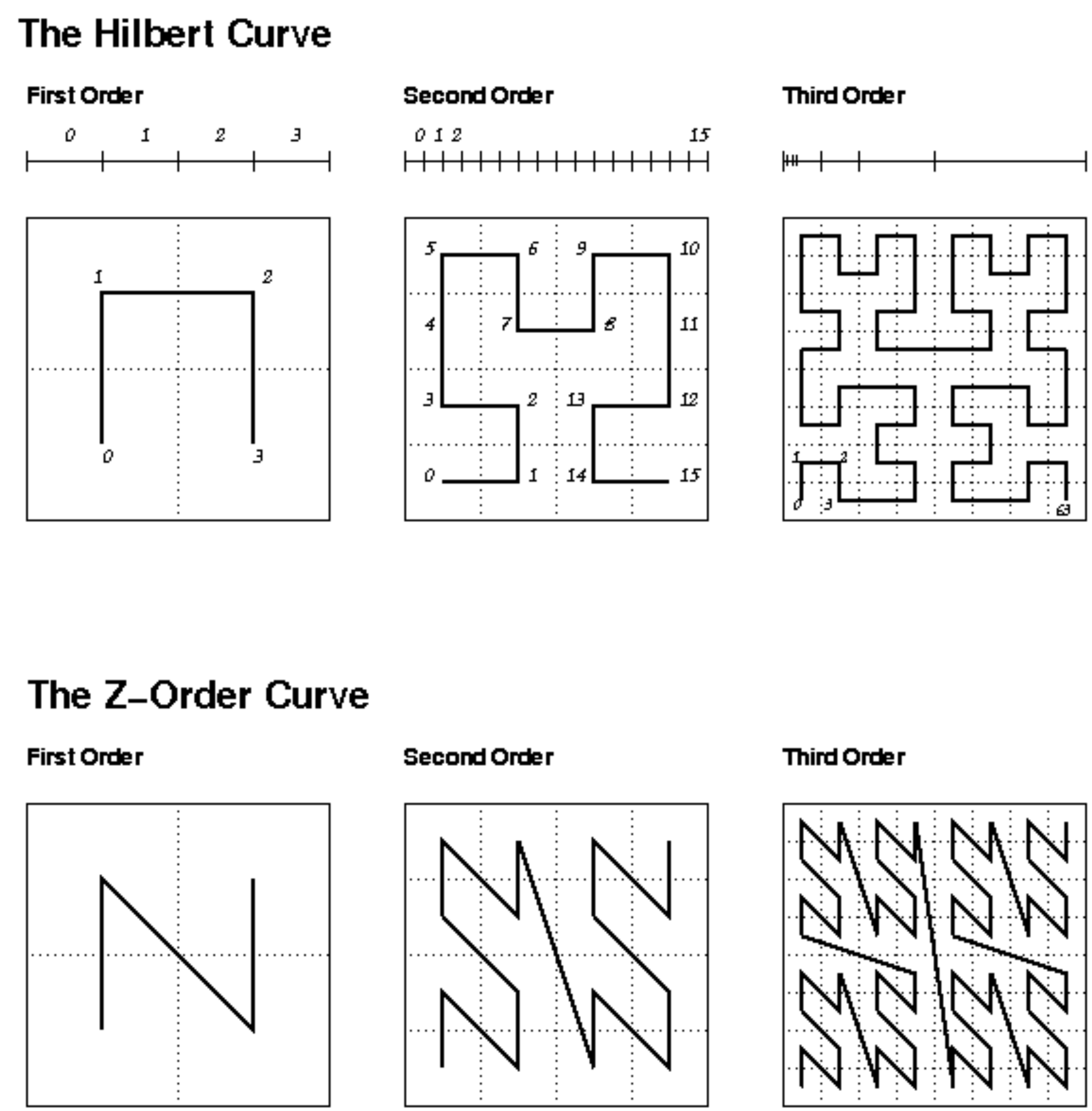
Global data comunications and synchronization

- Two most time-consuming in the FMM:
  - p2p — purely local
  - m2l — exhibits “hierarchical synchronization”



Load balancing

- FMM load-balanced
  - space-filling curves: Morton, Hilbert
  - work-only (no comm)
- PetFMM:
  - graph-partitioning
  - will it scale?
  - hierarchical partition?



plan for an "ExaFMM"

1) our present FMM technology is state-of-the-art;
2) we possess the potential for a substantial performance hike

AND all our codes are always open!

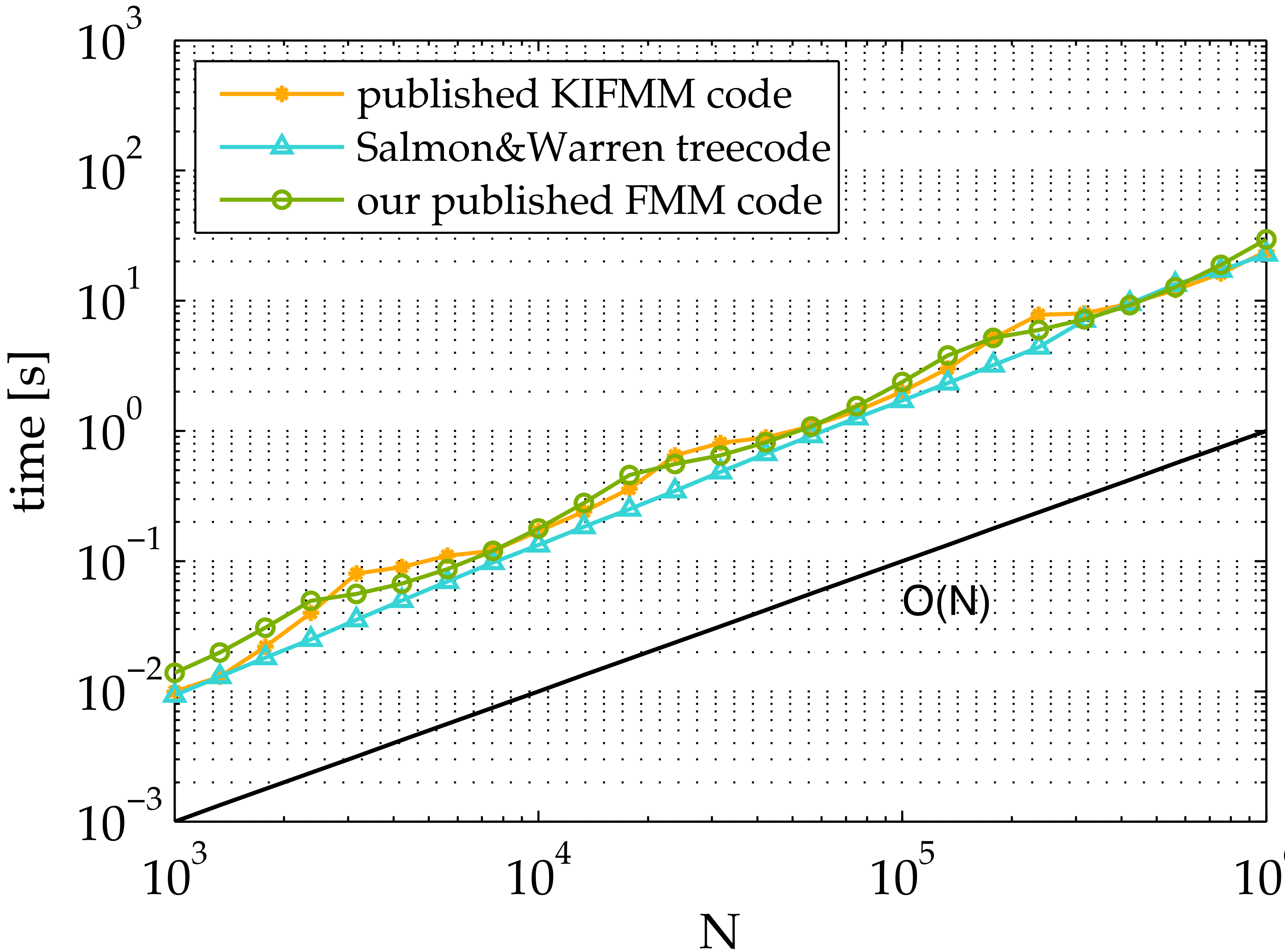
Present FMM state-of-the-art

Single-node performance:

timings of published `kifmm` *code (2006), S&W treecode (2000) and our code*

- equal performance
- same accuracy, measured L^2-norm error 10^{-3}

Single CPU core, Intel Core i7 2.67 (no SSE)



New experimental FMM with higher performance

Optimized code:

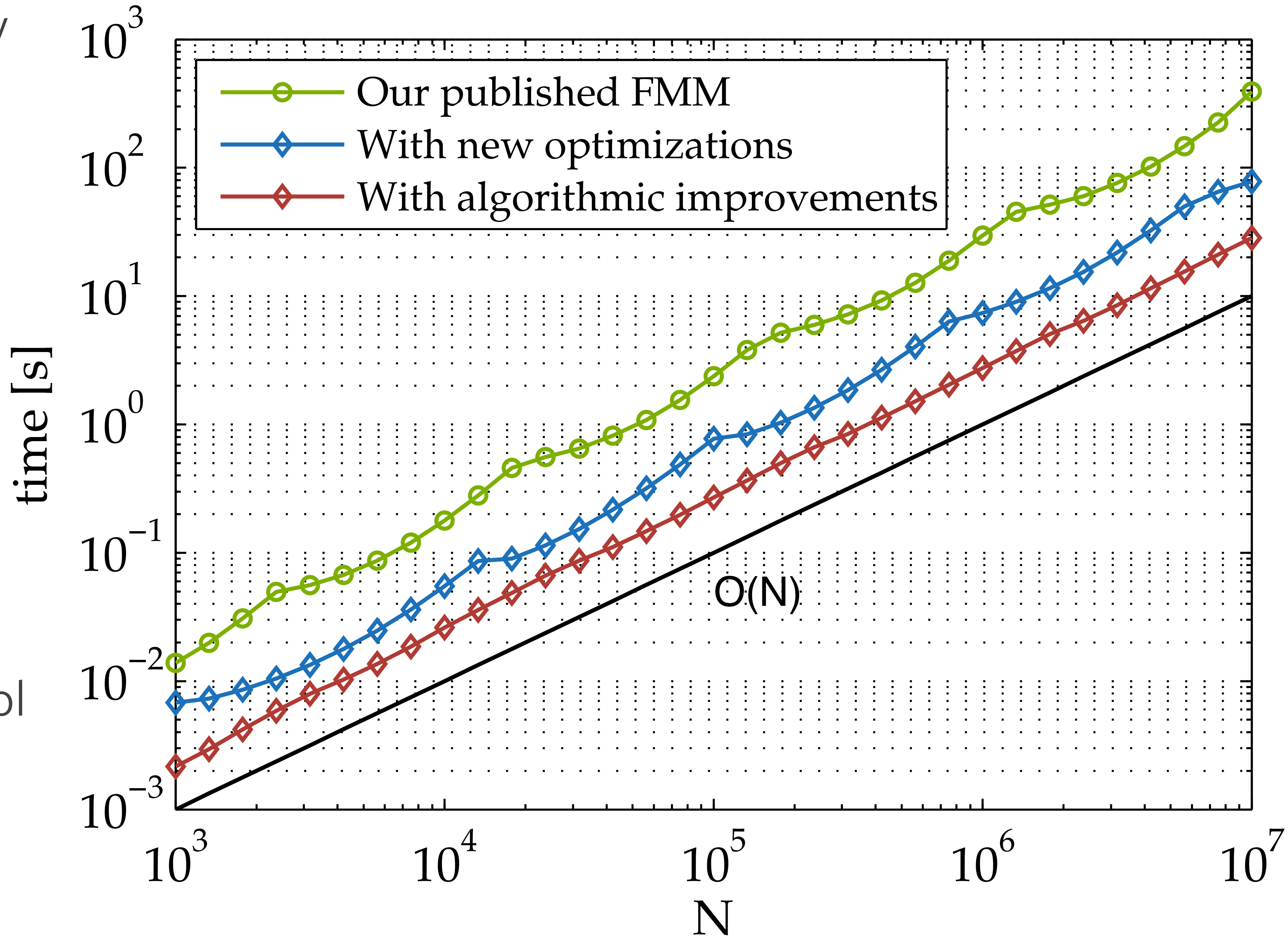
explicit inline assembly within the p2p kernel, implementing SIMD

- 5x speed-up, single precision

Algorithmic improvements:

i) hybridize FMM with treecode

ii) dynamic error-control



- other recent work

Optimizing and Tuning the Fast Multipole Method for State-of-the-Art Multicore Architectures

Aparna Chandramowlishwaran*†, Samuel Williams*, Leonid Oliker*, Ilya Lashuk†, George Biros†, Richard Vuduc†

**CRD, Lawrence Berkeley National Laboratory, Berkeley, CA 94720*
†College of Computing, Georgia Institute of Technology, Atlanta, GA

In *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, IEEE, pp. 1–12 (Atlanta, GA; April 2010)

Summary so far ...

- **PetFMM** — open library, dynamic load balancing, comm minimizing
  - open question: will strategy scale to 1000s procs? hierarchical partition?
- **Performance** on single node:
  - matching other s.o.t.a. codes
- **Algorithmic innovations**:
  - hybrid treecode/FMM
  - variable order/variable box-opening for minimum work to achieve target accuracy

But there is more ...

- Fault-tolerance:
  - traditional checkpointing no longer adequate by itself
    - instead: replicate threads, correctness checks on-the-fly
  - FMM allows natural correctness checks at the time of selecting p
- Autotuning the FMM:
  - natural: use tests/work estimats to select particles per box, p, and box-opening parameters.
  - parameter selection for load-balancing