Brief Announcement: Towards a Communication Optimal Fast Multipole Method and its Implications at Exascale

Aparna Chandramowlishwaran
Georgia Institute of Technology
aparna@gatech.edu

Jee Whan Choi
Georgia Institute of Technology
jee@gatech.edu

Kamesh Madduri
Pennsylvania State University
madduri@cse.psu.edu

Richard Vuduc
Georgia Institute of Technology
richie@cc.gatech.edu

ABSTRACT

This paper presents the first in-depth models for compute and memory costs of the kernel-independent fast multipole method (KIFMM). The fast multiple method (FMM) has asymptotically linear time complexity with a guaranteed approximation accuracy, making it an attractive candidate for a wide variety of particle system simulations on future exascale systems. This paper reports on three key advances. First, we present lower bounds on cache complexity for key phases of the FMM and use these bounds to derive analytical performance models. Secondly, using these models, we present results for choosing the optimal algorithmic tuning parameter. Lastly, we use these performance models to make predictions about FMM's scalability on possible exascale system configurations, based on current technology trends. Looking forward to exascale, we suggest that the FMM, though highly compute-bound on today's systems, could in fact become memory bound by 2020.

Categories and Subject Descriptors: F.2 [Analysis of Algorithms and Problem Complexity]: Numerical Algorithms and Problems
Keywords: Fast Multipole Method; Cache Complexity Analysis; Performance Modeling; Exascale

1 Introduction

We report on a new analysis of memory hierarchy communication for the fast multipole method (FMM) [6], which is widely regarded as one of the most significant algorithms in scientific computing [3]. For a particle simulation involving n interacting particles, which naïvely is a $\mathcal{O}\left(n^2\right)$ computation, the FMM performs a work-optimal $\mathcal{O}\left(n\right)$ operations with user-selectable accuracy guarantee.

Our analysis refines the estimates of the constants, normally ignored in traditional asymptotic analyses of the FMM, with calibration against our state-of-the-art implementation [4, 5]. The result is an analytical performance model with two important properties. First, the model predicts the optimal setting of one of the FMM's tuning parameters, which in practice had previously required manual experimentation to set. Secondly, since the analysis includes important high-level architectural parameters, such as last-level cache capacity, the resulting models can be used to estimate whether the FMM will scale or not on future architectural designs.

In fact, our model suggests a new kind of high-level analytical *co-design* of the algorithm and architecture. For instance, classical analyses of *balance* relate algorithmic properties, such as intensity (intrinsic ratio of useful ops to bytes transferred), to a processor's balance (its peak ops/sec divided by peak bandwidth). The total time T is given by the sum of the times taken by the phases of FMM. Based on the analytical execution time estimates for the most expensive phases of FMM, we derive analytically an optimal value for an algorithmic tuning parameter, q (Section 2), which practitioners had previously thought could only be determined experimentally.

$$q = \frac{\gamma^{3/2}}{C_1}\sqrt{C_2 + C_3\frac{C_0}{\beta_{mem}}} \quad (1)$$

The constants C_1, C_2, and C_3 can be estimated given a kernel and an implementation. The fraction $\frac{C_0}{\beta_{mem}}$ is the processor's balance and γ is the number of digits of precision required. For our current state-of-the-art multicore implementation, $\frac{C_2}{C_3} \approx 50$ and on a single socket Intel Westmere node $\frac{C_0}{\beta_{mem}} = 2.6$ resulting in the optimal $q \approx 250$ which exactly matches our experimental value.

If we further assume that $q = \mathcal{O}(\gamma^{\frac{3}{2}})$, then T can be simplified as the expression below.

$$T = \frac{N\gamma^{3/2}}{C_0}(C' + C''\frac{C_0}{\beta_{mem}}) \quad (2)$$

C' and C'' are constants defined in terms of the previous constants.

One corollary of our analysis is that the accuracy of the FMM and one of its algorithmic tuning parameters can be used to compensate for processor imbalance, which is an unavoidable technology trend. However, we also find that although the FMM is today compute-bound and therefore highly scalable in practice, the current trajectory of processor architecture design could cause the FMM to become communication-bound as early as the year 2020.

2 Fast Multipole Method

Given a system of N *source* particles, with positions given by $\{y_1, \ldots, y_N\}$, and N *targets* with positions $\{x_1, \ldots, x_N\}$, we wish to compute the N sums, $f(x_i) = \sum_{j=1}^{N} K(x_i, y_i) \cdot s(y_j)$, where $f(x)$ is the desired *potential* at target point x; $s(y)$ is the *density* at source point y; and $K(x, y)$ is an *interaction kernel* that specifies "the physics" of the problem. The FMM can *approximate* of all of these sums in an optimal $O(N)$ time with a guaranteed user-specified accuracy ϵ [6]. This acceleration is based on two key ideas: (i) organizing the points spatially in a *tree representation*, such as an octree in three dimensions or a quadtree in 2D; and (ii) *fast approximate evaluation*, in which we compute summaries at each node using a constant number of tree traversals with constant work per node.

The tree is constructed so that the leaves contain no more than q points each, where q is a tuning parameter chosen by the user. We also associate with each node of the tree one or more neighbor *lists*. Each list has bounded constant length and contains (logical) pointers to a subset of other tree nodes. These are canonically known as the U, V, W, and X lists. Given the tree T, evaluating the sums consists of six distinct computational phases: one per U, V, W, and X lists (which are all neighborhood iterations), as well as *upward* (up) and *downward* (down) phases.

Our analysis uses our own recent implementation for multicore and GPU systems [4, 5, 7, 9].

3 Analytical Performance Model for FMM

In this section, we present the lower bounds for the two key phases of FMM. We assume a uniform random distribution of source and target points for the rest of the analysis.

We assume a simple two-level memory hierarchy, consisting of an infinite memory and a cache of size Z. Data is transferred between the memory and cache in cache lines of size L.

3.1 Near field Interactions (U list step)

For each target leaf box, this phase of the FMM algorithm performs a direct summation of potentials due the source boxes in its immediate neighborhood. The neighborhood of a box B is defined to be the set of all the source leaf boxes adjacent to B, and contains B as well. This list of boxes L_U^B is called the U list, and we refer to this near field interactions evaluation phase as the U list step.

For each target-source pair, a dense matrix of kernel evaluations is created, and the target potential vector is updated with a dense matrix and vector multiplication. Given b leaf boxes and assuming q points per leaf box, the computational complexity of the near field interaction phase is $\mathcal{O}(bq^2)$. In 3D, the operation count is more precisely $27bq^2$. This estimate can be further refined to account for boundary boxes, and we have $(3b^{\frac{1}{3}} - 2)^3 q^2$.

The time spent performing floating-point operations in the U list step is the total number of floating point operations, divided by the peak computation throughput (C_0) in floating-point operations per unit time.

$$T_{comp,u} = \frac{C_u^1 \cdot (3b^{1/3} - 2)^3 \cdot q^2}{C_0} \tag{3}$$

C_u^1 is a kernel- and implementation-dependent constant.

To account for memory costs in accessing the source and target box data structures, we observe that the outer loops of the computation can be modeled as a sparse matrix vector multiply (SpMV). Each source box contains q points on an average. For each point, the position (x, y, and z coordinates) and density are maintained, resulting in a cumulative size of $4q$ machine words per source or target box.

Blelloch et al. [1, 2] present a cache-oblivious algorithm for SpMV that is based on a separator-based reordering of the matrix. They show that if the support graph of a matrix satisfies the n^ϵ edge-separator theorem [8], then such a matrix, when laid out in row-major format after reordering, would incur at most $\mathcal{O}\left(\frac{m}{L} + \frac{n}{Z^{1-\epsilon}}\right)$ (where m is the number of non-zeros in the matrix) cache misses for SpMV. The U list implicit dependency matrix is indeed structured, and this is a result of the spatial sorting of the boxes during tree construction. The source boxes are also stored contiguously in row-major format, and so we can adapt the SpMV bounds for near interaction computation.

The memory access costs for this step are comprised of read accesses to the source boxes, the U lists for each target box, and updates to the target leaf box potentials. The rows of the kernel matrix K are constructed on-the-fly for each source-target pair prior to matrix vector multiplication, and so we do not consider accesses to this matrix. The U list upper bounds for the number of cache lines fetched are as follows:

$$\begin{aligned}
Q_u &= Q_{u\_src} + Q_{u\_trg} + Q_{u\_lists} \\
Q_{u\_src} &\le k_u \cdot \frac{N}{q} \cdot \frac{4q}{L} = \frac{4k_u N}{L} \\
Q_{u\_trg} &\le \frac{N}{q} \cdot \frac{4q}{L} = \frac{4N}{L} \\
Q_{u\_lists} &\le \frac{k_u \frac{N}{q}}{L}
\end{aligned}$$

Here, k_u is the average number of source boxes in the U list of a target leaf box. The above bound for Q_{u_src} assumes that there is no reuse of source boxes. The cache complexity for U list is thus dominated by the time to read source boxes.

Utilizing the SpMV bounds from [1] and assuming $\epsilon = 2/3$ for 3D, we get a tighter bound on Q_{u_src}, and thus the overall cache complexity. Since each non-zero in the matrix corresponds to a source box of size $4q$, we scale the fast memory capacity Z by a factor of $4q$.

$$Q_u \le \frac{4N}{L} + \frac{k_u \frac{N}{q}}{L} + \frac{4N}{L} + \frac{\frac{N}{q}}{\left(\frac{Z}{4q}\right)^{\frac{1}{3}}} \tag{4}$$

The dominant memory access time in this step is modeled as the total data fetched into fast memory, divided by the peak rate at which data is fetched into memory (i.e., memory bandwidth β_{mem}).

$$T_{mem,u} = \frac{C_u^2 N}{\beta_{mem}} + \frac{C_u^3 NL}{\beta_{mem}(Z^{\frac{1}{3}} q^{\frac{2}{3}})} \tag{5}$$

C_u^2 and C_u^3 are implementation- and machine-dependent constants that we determine empirically by fitting the execution times to the model.

3.2 Far field Interactions (V list step)

For each target box in the tree, this phase accumulates the multipole expansions of the source boxes in its V list into a local expansion. This step is also called multipole to local (M2L) translation. The V list of a box B is defined to be the set of all source boxes that are children of the neighbors of box B's parent, but not adjacent to B itself. The computation performed in the V list is 3D convolution. We implement this in 3 steps, namely, (a) 3D FFT, (b) complex pointwise multiplication in the frequency domain, and (c) 3D inverse FFT. Assuming b_s source boxes, the computational complexity of the FFT phase is $\mathcal{O}(b_s \,.\, p^{\frac{3}{2}} \log p)$ where p is a constant determined by the desired accuracy ($p = \mathcal{O}(\gamma^2)$). The inverse FFT's are done once for each target box, resulting in a complexity of $\mathcal{O}(b_t \,.\, p^{\frac{3}{2}} \log p)$, assuming b_t target boxes. Each target box performs k_v pointwise multiplications ($k_v = 189$ for an interior box for an uniform distribution), and has an asymptotic complexity of $\mathcal{O}(b_t \,.\, k_v \,.\, p^{\frac{3}{2}})$.

Refining these estimates, the computational time for V list if given by

$$T_{comp,v} = \frac{C_v^1 \ (b_s + b_t + 343) \ p^{\frac{3}{2}} \log p}{C_0} + \frac{C_v^2 \ b_t \ k_v \, p^{\frac{3}{2}}}{C_0} \tag{6}$$

C_v^1 and C_v^2 are implementation-dependent constants.

The number of translation operators (316) is fixed and hence we assume they fit in the shared cache Z. Hence, the effective cache size becomes $Z^{'} = Z - 316p^{\frac{3}{2}}$. The V-list implicit dependency matrix between target boxes, source boxes, and translation operators is also structured and similar to U list, reapplying the SpMV bounds from [2], we get an upper bound on the cache complexity for this phase:

$$Q_v \le \frac{(b_t + b_s)\ p^{\frac{3}{2}}}{L} + \frac{k_v \ b_t}{L} + \frac{b_t}{\left(\frac{Z'}{p^{\frac{3}{2}}}\right)^{\frac{1}{3}}} \tag{7}$$

Considering the higher order terms, the memory access time of V list can be approximated by,

$$T_{mem,v} = \frac{C_v^3 N p^{\frac{3}{2}}}{q\beta_{mem}} + \frac{C_v^4 N p^{\frac{1}{2}} L}{(Z'^{\frac{1}{3}} q)\beta_{mem}} \tag{8}$$

4 Exascale Projections

Using the above analytic expression for execution time and the optimal choice of q, we predict the execution time for large-scale problem instances on possible future CPU-based exascale systems. The machine characteristics of the exascale system are based on extrapolating historical technology trends [10]. Figure 1 shows the execution time split into computational and memory access time. We observe that the crossover point when the memory access time T_{mem} matches the compute time T_{comp} would occur around 2020. We are currently extending our KIFMM performance model and analysis to GPU-based systems.

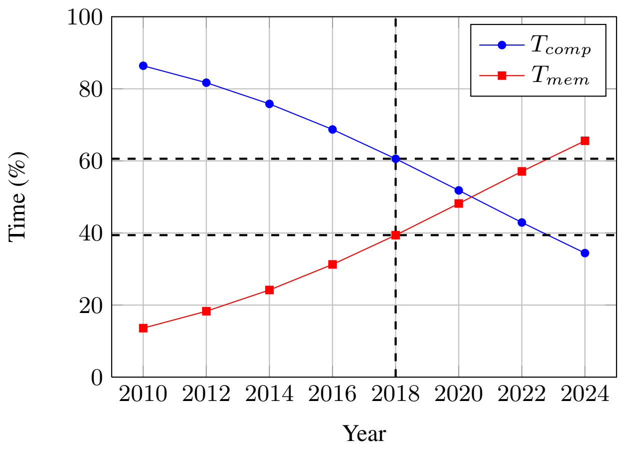

Figure 1: A depiction of the KIFMM computational and memory costs for parallel execution on extrapolated CPU-like multicore systems. The problem size N starts at 4 million points in 2010, and is scaled at the same rate as the cache size Z.

5 References

[1] G. E. Blelloch, R. A. Chowdhury, P. B. Gibbons, V. Ramachandran, S. Chen, and M. Kozuch. Provably good multicore cache performance for divide-and-conquer algorithms. In *Proc. Nineteenth Annual ACM-SIAM symposium on Discrete algorithms (SODA '08)*, pages 501–510, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[2] G. E. Blelloch, P. B. Gibbons, and H. V. Simhadri. Low depth cache-oblivious algorithms. In *Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Thira, Santorini, Greece, July 2010.

[3] J. Board and K. Schulten. The fast multipole algorithm. *Computing in Science and Engineering*, 2(1):76–79, January/February 2000.

[4] A. Chandramowlishwaran, K. Madduri, and R. Vuduc. Diagnosis, tuning, and redesign for multicore performance: A case study of the fast multipole method. In *Proc. ACM/IEEE Conf. Supercomputing (SC)*, New Orleans, LA, USA, November 2010.

[5] A. Chandramowlishwaran, S. Williams, L. Oliker, I. Lashuk, G. Biros, and R. Vuduc. Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures. In *Proc. IEEE Int'l. Parallel and Distributed Processing Symp. (IPDPS)*, Atlanta, GA, USA, April 2010.

[6] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comp. Phys.*, 73:325–348, 1987.

[7] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros. A massively parallel adaptive fast multipole method on heterogeneous architectures. In *Proc. ACM/IEEE Conf. Supercomputing (SC)*, Portland, OR, USA, November 2009.

[8] R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2), 1979.

[9] A. Rahimian, I. Lashuk, D. Malhotra, A. Chandramowlishwaran, L. Moon, R. Sampath, A. Shringarpure, S. Veerapaneni, J. Vetter, R. Vuduc, D. Zorin, and G. Biros. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *Proc. ACM/IEEE Conf. Supercomputing (SC)*, New Orleans, LA, USA, November 2010.

[10] R. Vuduc and K. Czechowski. What GPU computing means for high-end systems. *IEEE Micro*, 31(4):74–78, July/August 2011.