Commun. Comput. Phys.
doi: 10.4208/cicp.230517.300617sw

Vol. **23**, No. 1, pp. 296-314
January 2018

COMPUTATIONAL SOFTWARE

Revision of DASHMM: Dynamic Adaptive System for Hierarchical Multipole Methods

J. DeBuhr*, B. Zhang and T. Sterling

Center for Research in Extreme Scale Technologies, School of Informatics and Computing, Indiana University, Bloomington, IN, 47404, USA.

Received 23 May 2017; Accepted (in revised version) 30 June 2017

Abstract. We present version 1.2.0 of DASHMM, a general library implementing hierarchical multipole methods using the asynchronous multi-tasking HPX-5 runtime system. Compared with the previous release [10], this new version: (1) enables execution in both shared and distributed memory architectures; (2) extends DASHMM's infrastructure to support advanced multipole methods [18]; and (3) provides built-in implementations of both the Yukawa [15] potential and Helmholtz [16] potential in the low frequency regime. These additions have not impacted the user interface, which remains simple and extensible.

AMS subject classifications: 15A06, 31C20, 68N19

Key words: Generic multipole methods, Laplace potential, Yukawa potential, Helmholtz potential, distributed computing.

Program summary

Program title: DASHMM version 1.2.0

Nature of problem: Evaluates the Laplace, Yukawa or Helmholtz potentials at N target locations induced by M source points in three dimensions.

Software license: BSD 3-Clause

CiCP scientific software URL: `http://www.global-sci.com/code/dashmm-1.2.0.tar.gz`

Distribution format: .gz

Programming language(s): C++

Computer platform: x86_64

*Corresponding author. *Email address:* `jdebuhr@indiana.edu` (J. DeBuhr)

Operating system: Linux

Compilers: GCC 4.8.4 or newer; icc (tested with 15.0.1)

RAM:

External routines/libraries: HPX-5 4.0.0 or later

Running time:

Restrictions:

1 Introduction

Hierarchical Multipole Methods (HMMs) are a key component of many science applications in a wide range of fields. However, conventional parallel programming practice leaves many of these applications strong-scaling constrained. Asynchronous Many-Tasking (AMT) runtime systems offer some promise as a means to overcome the scaling constraints of HMMs. Many such programming models and runtime systems are difficult to incorporate into existing code, leading to the need to rewrite potentially large applications, a cost which is often not feasible for many groups. What is needed is a system that can provide both ready-made and user-created HMMs, in a form that is easy to use, and which obviates the need to write application code that targets advanced, and often experimental, AMT systems. The Dynamic Adaptive System for Hierarchical Multipole Methods (DASHMM) is a scientific software library that provides easy-to-use, extensible, scalable and efficient parallel implementations of HMMs on both shared and distributed memory architectures.

This paper presents a major update to DASHMM version 0.5.0 [10]. The centerpiece of this update is the ability of the library to operate on distributed memory architectures. The implementation deviates from typical bulk-synchronous executions, which mostly rely on locally essential trees [24–26], in favor of adaptive runtime techniques. DASHMM also differs from previous adaptive runtime implementations of HMMs [2,3,13,22], which were limited to shared memory architectures. DASHMM maintains the simple interface of the earlier version that requires no particular knowledge of the advanced experimental runtime system that provides DASHMM's parallelism. Further, this parallelism requires no explicit specification by the user about where the data is to be placed across the system.

In addition, this update includes a new built-in method, and two new built-in kernels. DASHMM now includes an advanced FMM [7,18], which this paper refers to as the FMM97 method, that uses exponential expansions and the merge-and-shift technique. This method can be applied to the previously supplied Laplace kernel, as well as the two new built-in kernels, Yukawa and low-frequency Helmholtz. The Yukawa kernel is widely used in Brownian dynamics [21], and in the computation of non-bonded interactions [4]. The Helmholtz kernel is has wide applications in computational electromagnetics [8], the scattering of radiation [14] and electromagnetic compatibility/interference, the design of antennas, radar [5], optical and imaging systems, frequency-selective surfaces

and metamaterials [19]. With the addition of these kernels, and the FMM97 advanced method, DASHMM's set of built-in methods and kernels covers a wide range of application classes.

The organization of this paper is as follows. In Section 2, the details of the expansions for the three kernels provided with DASHMM are supplied. Conceptual extensions to DASHMM required to support the FMM97 method are given in Section 3. Details about the distributed operation of DASHMM are presented in Section 4. The results of a strong scaling study is given in Section 5. Finally, Section 6 concludes the paper.

2 Mathematical foundations

The previous release of DASHMM [10] implemented the classical Fast Multipole Method (FMM) [6, 17]. In three dimensions, each box b in this method needs to perform up to 189 multipole-to-local translations, which seriously impacts the performance. A major improvement was made to address this issue in 1997 [18]. The idea is to introduce a third expansion, the exponential expansion, such that the translation is a component-wise (or diagonal) operation. Additionally, it becomes much easier to explore the overlapping regions of adjacent boxes where multipole-to-local translations are required. This is referred as the *merge-and-shift* technique in the literature and can reduce the average number of translations from 189 down to 40. The FMM97 method was first applied to the Laplace potential [18], and then extended to the Yukawa potential [15], and Helmholtz potential in the low-frequency regime [16]. This section gathers the related mathematical foundations in one place, and supplements the missing formulations in existing publications.

In this section, the multipole and local expansions for the Laplace ($\frac{1}{r}$), Yukawa ($\frac{\pi e^{-\lambda r}}{2\lambda r}$), and Helmholtz ($\frac{e^{i\lambda r}}{r}$) potentials are of the form

$$\text{Laplace} \quad \sum_{n=0}^{\infty}\sum_{m=-n}^{n} \frac{M_n^m}{r^{n+1}} Y_n^m(\theta,\phi), \quad \sum_{n=0}^{\infty}\sum_{m=-n}^{n} L_n^m r^n Y_n^m(\theta,\phi), \tag{2.1}$$

$$\text{Yukawa} \quad \sum_{n=0}^{\infty}\sum_{m=-n}^{n} M_n^m k_n(\lambda r) P_n^{|m|}(\cos\theta) e^{im\phi}, \quad \sum_{n=0}^{\infty}\sum_{m=-n}^{n} L_n^m i_n(\lambda r) P_n^{|m|}(\cos\theta) e^{im\phi}, \tag{2.2}$$

$$\text{Helmholtz} \quad \sum_{n=0}^{\infty}\sum_{m=-n}^{n} M_n^m h_n(\lambda r) P_n^{|m|}(\cos\theta) e^{im\phi}, \quad \sum_{n=0}^{\infty}\sum_{m=-n}^{n} L_n^m P_n^{|m|}(\cos\theta) e^{im\phi} j_n(\lambda r), \tag{2.3}$$

where $\{M_n^m\}$ represent multipole moments, and $\{L_n^m\}$ represent local expansion coefficients. The spherical harmonics are given by

$$Y_n^m(\theta,\phi) = \sqrt{\frac{(n-|m|)!}{(n+|m|)!}} P_n^{|m|}(\cos\theta) e^{im\phi},$$

i_n and k_n are the modified spherical Bessel functions, h_n is the spherical Hankel function of the first kind, and j_n is the spherical Bessel function of the first kind.

In the following discussion, the origin of the coordinates is at the center of the box for which the multipole expansion is being translated. The formulas presented directly apply to boxes that lie along the $+z$-axis. Boxes along the $-z$-axis can be easily handled using reflection through the origin. For boxes along x/y-axis, the coordinates need to be first rotated such that the old x/y-axis is the $+z$-axis in the rotated frame.

2.1 Laplace

The spherical harmonics $Y_n^m(\theta,\phi)$ connects to the partial derivatives as follows

$$\frac{Y_n^0(\theta,\phi)}{r^{n+1}}=A_n^0\partial_z^n(1/r),\quad \frac{Y_n^m(\theta,\phi)}{r^{n+1}}=A_n^m\partial_+^m\partial_z^{n-m}(1/r),\quad \frac{Y_n^{-m}(\theta,\phi)}{r^{n+1}}=A_n^m\partial_-^m\partial_z^{n-m}(1/r),$$

where

$$\partial_+=\partial_x+i\partial_y,\quad \partial_-=\partial_x-i\partial_y,\quad A_n^m=\frac{(-1)^{n+m}}{\sqrt{(n-m)!(n+m)!}}.$$

Apply formula [23, p. 1256]

$$\frac{1}{r}=\frac{1}{2\pi}\int_0^\infty e^{-uz}\int_0^{2\pi}e^{iu(x\cos\alpha+y\sin\alpha)}d\alpha du,\quad z>0.$$

Some algebraic work gives

$$\partial_z^n(1/r)=\frac{1}{2\pi}\int_0^\infty e^{-uz}(-u)^n\int_0^{2\pi}e^{iu(x\cos\alpha+y\sin\alpha)}d\alpha du,$$
$$\partial_z^{n-m}\partial_\pm^m(1/r)=\frac{1}{2\pi}\int_0^\infty e^{-uz}(-u)^{n-m}\int_0^{2\pi}e^{iu(x\cos\alpha+y\sin\alpha)}i^m e^{\pm im\alpha}u^m d\alpha du.$$

Substitute these results into (2.1), the multipole expansion can be rearranged as

$$\sum_{n=0}^\infty\sum_{m=-n}^n\frac{M_n^m}{\sqrt{(n-m)!(n+m)!}}\frac{1}{2\pi}\int_0^\infty e^{-uz}u^n\int_0^{2\pi}e^{iu(x\cos\alpha+y\sin\alpha)}i^{|m|}e^{im\alpha}d\alpha du.$$

The outer integral is handled by generalized Gaussian quadrature [28]. For accuracy requirement ϵ, the number of quadrature points is denoted $S(\epsilon)$, and the quadrature nodes and weights are given by $\{u_k,w_k\}$, $k=1,\cdots,S(\epsilon)$. For each u_k, the inner integral is handled by the trapezoid rule using $M(k)$ quadrature points. The multipole expansion can then be expressed as

$$\sum_{k=1}^{S(\epsilon)}\sum_{j=1}^{M(k)}W(k,j)e^{-u_kz}e^{iu_k(x\cos\alpha_{k,j}+y\sin\alpha_{k,j})},\tag{2.4}$$

where

$$W(k,j)=\frac{w_k}{M(k)}\sum_{m=-\infty}^\infty i^{|m|}e^{im\alpha_{k,j}}\sum_{n=|m|}^\infty\frac{M_n^m}{\sqrt{(n-m)!(n+m)!}}u_k^n,\quad \alpha_{k,j}=\frac{2j\pi}{M(k)}.\tag{2.5}$$

Eq. (2.5) translates the multipole expansion into the exponential expansion (2.4). Notice that (2.5) can be interpreted as evaluating a Fourier series expansion at $\alpha_{k,j}$ where the inner sum over n is the Fourier coefficient.

To convert (2.4) into a local expansion where the original expansion is valid, one applies the formula [20, p. 123]

$$(z+ix\cos\alpha+iy\sin\alpha)^n=r^n\left\{P_n(\cos\theta)+2\sum_{m=1}^{n}i^m\frac{n!}{(n+m)!}(-1)^mP_n^m(\cos\theta)\cos m(\phi-\alpha)\right\}$$

on each term of the Taylor expansion of $e^{-uz}e^{i(x\cos\alpha+y\sin\alpha)}$ and, after some algebraic work, finds

$$L_n^m=\frac{i^{|m|}}{\sqrt{(n-m)!(n+m)!}}\sum_{k=1}^{S(\epsilon)}(-u_k)^n\sum_{j=1}^{M(k)}W(k,j)e^{-im\alpha_{k,j}}. \tag{2.6}$$

2.2 Yukawa

The work for Yukawa potential relies on the integral representation [23]

$$\frac{\pi}{2}\frac{e^{-\lambda r}}{\lambda r}=\frac{1}{4\lambda}\int_0^\infty e^{-(u+\lambda)z}\int_0^{2\pi}e^{i\sqrt{u^2+2u\lambda}(x\cos\alpha+y\sin\alpha)}d\alpha d\lambda$$

and the use of $F(r^2)=k_0(\lambda r)$ and $f_n=r^nP_n^{|m|}(\cos\theta)e^{im\phi}$ in the formula [20, p. 126]

$$\begin{aligned}&f_n(\partial_1,\cdots,\partial_p)F(r^2)\\&=\left\{2^n\frac{d^nF}{d(r^2)^n}+\frac{2^{n-2}}{1!}\frac{d^{n-1}F}{d(r^2)^{n-1}}\nabla^2+\cdots+\frac{2^{n-2m}}{m!}\frac{d^{n-m}F}{d(r^2)^{n-m}}\nabla^{2m}+\cdots\right\}f_n(x_1,\cdots,x_p),\end{aligned} \tag{2.7}$$

yielding

$$k_n(\lambda r)P_n^{|m|}(\cos\theta)e^{im\phi}=\frac{1}{4\lambda}\int_0^\infty e^{-(\lambda+u)z}\int_0^{2\pi}e^{i\sqrt{u^2+2u\lambda}(x\cos\alpha+y\sin\alpha)}i^{|m|}P_n^{|m|}\Big(\frac{\lambda+u}{\lambda}\Big)e^{im\alpha}d\alpha du.$$

Substituting the result into (2.2) and discretizing the integral in the same fashion as Section 2.1, one arrives at

$$\sum_{k=1}^{S(\epsilon)}\sum_{j=1}^{M(k)}W(k,j)e^{-(u_k+\lambda)z}e^{i\sqrt{u_k^2+2u_k\lambda}(x\cos\alpha_{k,j}+y\sin\alpha_{k,j})}, \tag{2.8}$$

where

$$W(k,j)=\sum_{m=-\infty}^{\infty}e^{im\alpha_{k,j}}i^{|m|}\frac{\pi w_k}{2\lambda M(k)}\sum_{n=|m|}^{\infty}M_n^mP_n^{|m|}\Big(\frac{\lambda+u_k}{\lambda}\Big), \tag{2.9}$$

and $S(\epsilon)$, $M(k)$, $\{u_k,w_k\}$, $\alpha_{k,j}$ are defined similarly as in Section 2.1.

To convert the exponential expansion into a local expansion in the region where the original multipole expansion is valid, one could interpret each exponent

$$(u_k+\lambda)z-i\sqrt{u_k^2+2u_k\lambda}(x\cos\alpha+y\sin\alpha)$$

as the dot product of vector (x,y,z) and (x',y',z') whose spherical coordinates are given by $(x,y,z)=(r,\theta,\phi)$ and $(x',y',z')=(r',\theta',\phi')$. This gives $r'=\lambda$. Applying the Addition Theorem [1, Eq. 10.2.37], one has

$$\begin{aligned}&e^{-(u_k+\lambda)z}e^{i\sqrt{u_k^2+2u_k\lambda}(x\cos\alpha+y\sin\alpha)}\\&=\sum_{n=0}^{\infty}\sum_{m=-n}^{n}(2n+1)\frac{(n-|m|)!}{(n+|m|)!}(-1)^nP_n^{|m|}\Big(\frac{u_k+\lambda}{\lambda}\Big)e^{-im\alpha_j}i^{|m|}i_n(\lambda r)P_n^{|m|}(\cos\theta)e^{im\phi},\end{aligned}$$

which after some more algebraic work gives

$$L_n^m=(2n+1)\frac{(n-|m|)!}{(n+|m|)!}(-1)^ni^{|m|}\sum_{k=1}^{S(\epsilon)}P_n^{|m|}\Big(\frac{u_k+\lambda}{\lambda}\Big)\sum_{j=1}^{M(k)}W(k,j)e^{-im\alpha_j}. \tag{2.10}$$

2.3 Helmholtz

The Helmholtz potential starts with the integral representation [16]

$$\frac{e^{i\lambda r}}{r}=\frac{1}{2\pi}\int_0^{\infty}e^{-\sqrt{u^2-\lambda^2}z}\int_0^{2\pi}e^{iu(x\cos\alpha+y\sin\alpha)}\frac{u}{\sqrt{u^2-\lambda^2}}d\alpha du.$$

The outer integral behaves differently in the ranges $[0,\lambda]$ and $[\lambda,\infty]$, which are referred as propagating and evanescent waves, respectively. Next, choosing $F(r)=h_0(\omega r)$, $f_n=r^nP_n^{|m|}(\cos\theta)e^{im\phi}$ in (2.7), each term of the multipole expansion becomes

$$\begin{aligned}&h_n(\lambda r)P_n^{|m|}(\cos\theta)e^{im\phi}\\&=\frac{-i}{2\pi\lambda}\int_0^{\infty}e^{-\sqrt{u^2-\lambda^2}z}\int_0^{2\pi}e^{iu(x\cos\alpha+y\sin\alpha)}\frac{u}{\sqrt{u^2-\lambda^2}}P_n^{|m|}\Big(\frac{i\sqrt{u^2-\lambda^2}}{\lambda}\Big)e^{im\alpha}(-i)^nd\alpha du.\end{aligned}$$

For the propagating part, change variable by $u=\lambda\sin\theta$, and the integral becomes

$$\frac{1}{2\pi}\int_0^{\pi/2}e^{i\lambda\cos\theta z}\int_0^{2\pi}e^{i\lambda\sin\theta(x\cos\alpha+y\sin\alpha)}\sin\theta P_n^{|m|}(\cos\theta)e^{im\alpha}(-i)^nd\theta d\alpha.$$

Discretizing the inner integral using the trapezoid rule and the outer integral using Gauss-Legendre quadrature on the interval $[0,\pi/2]$, one has

$$\sum_{k=1}^{S_P(\epsilon)}\sum_{j=1}^{M_P(k)}W_P(k,j)e^{i\lambda\cos\theta_kz}e^{i\lambda\sin\theta_k(\cos\alpha_{k,j}x+\sin\alpha_{k,j}y)}, \tag{2.11}$$

where

$$W_P(k,j)=\sum_{m=-\infty}^{\infty}\frac{\sin\theta_k v_k}{M_P(k)}e^{im\alpha_{k,j}}\sum_{n=|m|}^{\infty}M_n^m P_n^{|m|}(\cos\theta_k)(-i)^n. \tag{2.12}$$

For the evanescent part, change variables by $\sigma^2=u^2-\lambda^2$, the integral becomes

$$\frac{-i}{2\pi\lambda}\int_0^{\infty}e^{-\sigma z}\int_0^{2\pi}e^{i\sqrt{\sigma^2+\lambda^2}(x\cos\alpha+y\sin\alpha)}P_n^{|m|}\Big(\frac{i\sigma}{\lambda}\Big)e^{im\alpha}(-i)^n d\alpha d\sigma.$$

Discretizing the inner integral using the trapezoidal rule and the outer integral using generalized Gaussian quadrature, one has

$$\sum_{k=1}^{S_E}e^{-\sigma_k z}\sum_{j=1}^{M_E(k)}e^{i\sqrt{\sigma_k^2+\lambda^2}(x\cos\alpha_{k,j}+y\sin\alpha_{k,j})}W_E(k,j), \tag{2.13}$$

where

$$W_E(k,j)=\sum_{m=-\infty}^{\infty}e^{im\alpha_{k,j}}\frac{u_k}{\lambda M_E(k)}\sum_{n=|m|}^{\infty}M_n^m P_n^{|m|}\Big(\frac{i\sigma_k}{\lambda}\Big)(-i)^{n+1}. \tag{2.14}$$

Notations such as $S_.(\epsilon)$ and $M_.(k)$ have the same meaning as those in the previous sections.

The exponential expansions can be converted back into a local expansion using techniques similar to that in Section 2.2 and the Addition Theorem [1, 10.1.47]. Each term of the local expansion L_n^m accumulates contributions from the propagating wave $L_{n,P}^m$ and the evanescent wave $L_{n,E}^m$, where

$$L_{n,P}^m=(2n+1)i^n\frac{(n-|m|)!}{(n+|m|)!}\sum_{k=1}^{S_P}P_n^{|m|}(\cos\theta_k)\sum_{j=1}^{M_P(k)}e^{-im\alpha_{k,j}}W_P(k,j), \tag{2.15}$$

$$L_{n,E}^m=(2n+1)i^n\frac{(n-|m|)!}{(n+|m|)!}\sum_{k=1}^{S_E}P_n^{|m|}\Big(\frac{i\sigma_k}{\lambda}\Big)\sum_{j=1}^{M_E(k)}e^{-im\alpha_{k,j}}W_E(k,j). \tag{2.16}$$

3 Extensions supporting FMM97

The addition of support for the FMM97 method required two extensions to the existing DASHMM infrastructure.

First, DASHMM now sorts expansions into *normal* and *intermediate* types. A normal expansion includes the common multipole or local expansion, while an intermediate expansion is used to reduce the M→L operation costs. The exponential expansions in Section 2 is an exemplar of an intermediate expansion. The intermediate expansion introduces three additional operators to DASHMM: (1) M→I that translates a multipole expansion into an intermediate expansion; (2) I→I that shifts the center of intermediate expansion; and (3) I→L that translates the intermediate expansion back into a local

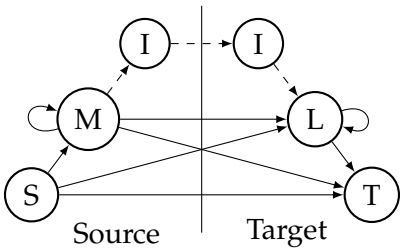

Figure 1: Diagram of translation operators in DASHMM. Basic multipole methods use multipole (M) and local (L) expansions, and eight operators (shown in solid lines) that connect them to the sources (S) and targets (T). Advanced multipole methods use intermediate expansions (I) and three additional operators (shown in dashed lines). The M→L operator is decomposed into a chain of M→I, I→I, and I→L operations in advanced multipole methods.

expansion. Unlike the M→L operator that occurs only between boxes of the same hierarchy level in classical FMM, I→I can happen across different levels, as demonstrated in the merge-and-shift techniques adopted in the FMM97 method. Fig. 1 shows the set of possible operators supported by DASHMM both before (solid arrows) and after (dashed arrows) this addition.

Second, the inclusion of intermediate expansion types in DASHMM lead to the need to make a distinction between the mathematical concept of an expansion, and the concept as used in DASHMM. The mathematical concept is demonstrated in Section 2, and is the expansion of some potential in a certain manner, such as with spherical harmonics. The concept of an expansion in DASHMM is wider: each expansion in DASHMM can contain multiple mathematical expansions. The prototypical case for this is for the FMM97 method. Each box in the source tree will have six versions of the exponential expansion, one each for (+z, -z, +y, -y, +x, -x). These six mathematical expansions represent different versions, or *views*, of the same information; they are combined into a single DASHMM expansion. Thus, an expansion in DASHMM is a collection of `Views`, each of which is the information for a single mathematical expansion. The set of `Views` are related in some way, typically as equivalent representations of the same underlying information.

By extending the notion of expansion in DASHMM in this way, it adds the flexibility to not only handle advanced HMMs, such as FMM97, but also allows for a user to perform more than one HMM at the same time. In this use, each `View` would be an expansion for a different kernel, or a different interaction.

4 Distributed implementation

DASHMM 1.2.0 provides a parallel implementation of HMMs on distributed memory architectures using the HPX-5 runtime system. This section covers the details of this distributed evaluation. The following summarizes some common HPX terms used in this section. The HPX-5 runtime system provides a global shared memory space abstraction. A parcel is a form of active message that contains a description of the action to

be performed, argument data, and continuation information and is sent to the global address on which the action is to be perform. Program data and control dependencies are represented in memory by local control objects (LCOs). An LCO is an event-driven, lightweight, globally addressable synchronization object that co-locates data and control information. For more details on HPX-5 concepts, and how they relate to DASHMM, see [10].

4.1 Overview

Any given DASHMM evaluation occurs in a set of four distinct phases. First, the source and target data are distributed and a dual tree is constructed. Second, a specified HMM is used to traverse the dual tree and generate an explicit version of the directed acyclic graph (DAG) representing the computation. Third, this explicit DAG is instantiated into an implicit DAG formed of runtime objects that manage the parallel computation. Fourth, the execution represented by the runtime objects is carried out. The following subsections cover each stage in detail.

4.2 Dual Tree construction

DASHMM allows the sources of the interaction and the target locations to be distinct, and so DASHMM uses a Dual Tree. A dual tree is a pair of hierarchical space partitions, one for the source locations, and one for the target locations. This flexibility was chosen so that DASHMM can be applied in the widest possible context. The reasons for this choice are discussed in [29].

In DASHMM, the source and target positions are provided by the user. The library makes no assumptions about the initial distribution of these data across localities. Depending on how the user has supplied the data, DASHMM must be ready for the data to have an arbitrary distribution, including all on a single locality, or equally spread across all localities.

The construction of the dual tree begins with a determination of the domain of the problem, which is the smallest cubical volume that completely encompasses all the source and target locations. This reduction is first accomplished on each locality, and those results are then reduced across localities. The resulting volume gives the volume of the root of both the source and target trees. That the root of each tree has the same volume is not required of a Dual Tree, but it does simplify certain implementation details. In particular, instead of using floating point representations of the volumes of the various boxes in either tree, an index composed of integral values can give the volume of any box provided the overall domain is known. This avoids any potential issues with comparing finite precision floating point numbers.

After the overall domain is determined, the source and target data are distributed across the localities in the system. This distribution aims to place an equal number of sources and targets on each locality. Note that the source and targets are not distributed

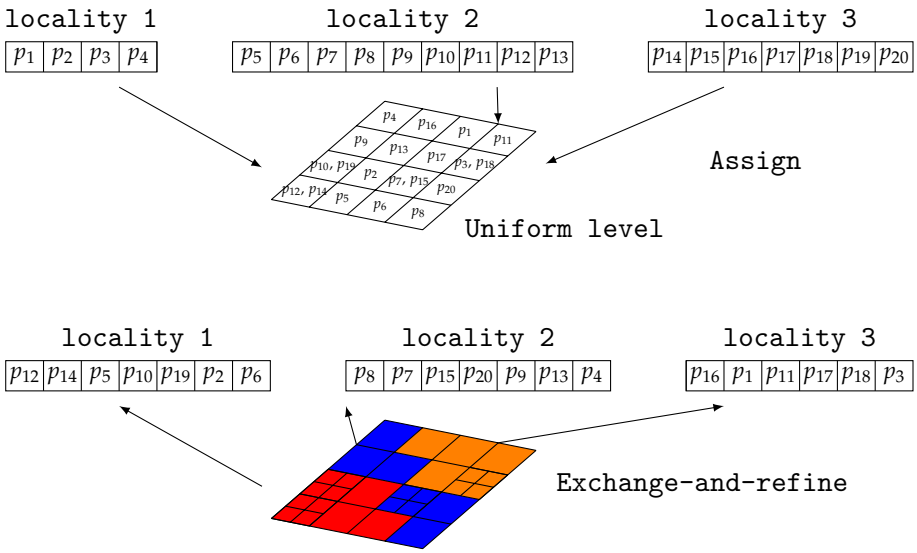

Figure 2: The tree is constructed from point data $\{p_i\}$, spread over the available localities. First, the points are mapped into a uniform refinement level. In the figure, this level is 2; the root box has been subdivided twice. Second, the set of boxes at the uniform level are partitioned among the localities using a Morton curve. In this example locality one is assigned 5 boxes at the uniform level, locality two is assigned 6, and locality three is assigned 5. Third, each locality sends points that it currently owns that have been assigned to other localities to those localities. Finally, once all the points for a given uniform level box have arrived, the tree below is constructed.

individually, but rather the collected source and target sets are distributed uniformly. To divide the data, the locations are mapped into a fixed level of the tree, called the *uniform refinement level*. Effectively, the top of the source and target trees are refined uniformly to a given level, partitioning the source and target data in the process. Then, the boxes at the uniform refinement level are divided among the available localities. The uniform refinement level is chosen to allow multiple boxes per locality on average. DASHMM selects:

$$\ell = \lceil \log_8 n_l \rceil + 1, \tag{4.1}$$

where ℓ is the uniform refinement level, and n_l is the number of localities available. See Fig. 2 for an example of this process for a two dimensional problem.

Once the occupations in each uniform level box is computed on each locality, these values are reduced across localities to have the total number of source and target locations in each box of the uniform level. To perform the distribution of the boxes, they are placed on a space filling curve, which is then segmented in a way that each segment contains roughly the same amount of data. DASHMM currently uses a Morton curve for this distribution. Once the owning locality of each uniform level box is known, the localities send any source or target data needed to its assigned owner.

Once a given locality has received all the source and target data for a given uniform level box, it can partition that branch of the dual tree. After each branch for a given

locality is completed, the result is a tree that contains all possible boxes up to the uniform refinement level, and the descendant branches of a subset of the boxes at the uniform refinement level. Each locality produces a similar structure, leading to a copy of the top of the dual tree on each locality and a single instance of any branch over all the available localities.

To aid in the construction of the DAG representing the work of the selected HMM, each locality then shares its branches with all other localities by sending a compressed representation of the branch's structure. This compressed representation is expanded into a full instance of the branch on each locality. Eventually, each locality has a full copy of the entire tree, though it only owns a subset of the branches of the tree. The slight excess in memory use pays for a much simpler discovery and distribution of the explicit DAG (see below).

DASHMM makes use of HPX-5 during this phase of tree construction in two ways. The first is that any reductions are performed using LCOs. Second, the various tasks that are performed in this procedure are instantiated as HPX-5 parcels, which can easily allow for asynchronous execution of these tasks. For instance, on a given locality, the construction of a given branch needs to depend only on all of the data for that branch arriving. So instead of requiring all of the source and target data communication to be complete across the system, a locality can launch the action that will partition a branch as soon as the data is fully available. This allows DASHMM, for instance, to hide messaging latency behind other work.

4.3 Explicit DAG

Once the dual tree is constructed, DASHMM then builds an explicit DAG, which represents the work of the selected HMM for the given source and target data. We add the qualifier 'explicit' to indicate that this representation is in a set of objects managed by the library. This is by contrast to the implicit DAG (see below) which is composed of objects managed by the HPX-5 runtime system. This stage of the evaluation is essentially a traversal of the dual tree [12, 27].

For a method to be usable by DASHMM, it must provide five operations that are called by the library. The exact details of these operations are not the concern of DASHMM, but are instead the concern of the implementer of the specific method. That being said, DASHMM is distributed with a set of common methods that can be used immediately. Four of the operations represent the various classes of transformations that are made to the source data along the path to the target points. The fifth operations allows an advanced method to react to the situation of the dual tree at runtime to avoid continued traversal of the dual tree in situations where an early termination might be to the benefit of performance, or some other characteristic. The following covers each operation in turn, giving not only their typical effect, but also the manner in which the operation is called.

The first operation that must be provided is *generate*. This operation is called at the

leaves of the source tree, and is responsible for creating the first summaries of the source data. Often all that *generate* will do is add an S→M edge to the DAG.

In *aggregate*, the summaries computed at a box's children are combined into a summary that applies for the box itself. This operation is called for each internal box of the source tree. Typically *aggregate* creates a set of M→M links from expansions for this box's children to the expansion for this box.

The *inherit* operation passes information from a box of the target tree to its children. This will occur for any box of the target tree, and typically involves the addition of L→L edges to the DAG.

In *process*, the links between the source and target trees are created. It is in this operation that the bulk of the differences between the various methods lie. This operation is called for each box of the target tree, but unlike the previous operations, this operation is also given a list of source tree boxes that are under consideration for interacting with this target box. The result of this operation is to either add an edge to the DAG connecting a box in the consider list and the box on which the operation is called, or to modify in some way the consider list, which is then passed into the process operation for this box's children. Typical edges added to the DAG during *process* are S→T, M→L, M→T, S→L, to name a few.

Finally, the *refinement test* operation is used to allow a method to have an early exit of the traversal. In some advanced methods, it can be advantageous to consider an internal box of the target tree as if it were really a leaf. This operation is called for each box of the target tree.

To actually perform the work of applying these operations, DASHMM makes use of HPX-5's task based description of execution. These tasks occur in two passes, one up the source tree, and one down the target tree. Each task in the source tree is the same, but is applied to different data, and each task in the target tree is the same, but is applied to different data. To start each pass of the traversal, the appropriate task is spawned at the root of each tree. To be sure of the existence of any source tree related DAG boxes, the traversal of the source tree is completed first, followed by the traversal of the target tree. In both cases, there is sufficient parallelism that waiting for the completion of the first pass does not significantly impact the utilization of the computational resources.

The task in the source tree first checks to see if the box to which it is applied is a leaf. If so, the *generate* operation is called. Otherwise, the same task is spawned one each for each child of the current box. When those tasks are complete, the *aggregate* operation is called on the current box. In this way, a tree of tasks are spawned that ultimately reach the leaves of the source tree. At the leaves, multipole moments are generated, which are then aggregated as the tasks waiting at the internal boxes are able to be completed.

The task in the target tree first performs the *refinement test*. Then, the *inherit* operation is performed to translate any information from the current box's parent (if it exists) to the current box. Then, in *process*, links are formed between the source and target trees. In this process the list of boxes that are being considered may have been modified. Finally, if the box is to be refined further, an identical task is spawned for each child of the current box,

passing in the new list of source tree boxes to consider.

Each available locality in the system has a complete copy of the structure of the dual tree, even if that locality owns only a fraction of the source and target data (see above). This means that each locality can generate this explicit DAG without coordination with the other localities. This removes any barriers to progress during this phase of the evaluation. The cost of this is that the DAG is replicated in memory at each locality. This design decision was made primarily so that DASHMM's development could focus on the execution of the DAG in parallel using asynchronous many-tasking. This approach, as can be seen in Section 5 has been successful. That being said, this is one place that will begin to impose scaling constraints as the time it takes to generate the DAG become larger than the time to execute a given localities portion of that DAG. This will be remedied in future versions of DASHMM.

4.4 Implicit DAG

Once the explicit DAG is constructed, the work represented by the DAG can be distributed and instantiated as runtime objects in preparation for the execution of the work. The main data being operated upon during an evaluation is the set of expansions at the various source and target boxes. These expansions often have multiple contributions, and because DASHMM makes use of asynchronous many-tasking, these contributions come at times that are unknown and potentially overlapping. To manage this contention, DASHMM uses LCOs to represent each expansion that must be computed. The implicit DAG is composed of the set of Expansion LCO objects. These not only provide access to the expansion data, but manage the concurrent modification of that data, and the continuation of execution once an expansion is calculated completely.

In HPX-5, all actions target a particular address in the global address space. By judiciously choosing on which localities to place the objects in a particular HPX-5 application, one controls the distribution of the work executed by the system. In the case of DASHMM, the placement of the expansion LCOs will determine the distribution of the work during execution. To this end, DASHMM uses a *distribution policy* to select which locality should own the nodes of the DAG. This policy operates on the explicit DAG, and can be different for different methods. It is simple to change to another policy that is provided with DASHMM, and relatively simple to create a new policy that might allow DASHMM to benefit from expert knowledge for a particular use-case.

In the same way that the explicit DAG is constructed on each locality, the distribution of DAG nodes is computed on each locality. The output of this distribution is an assignment of each DAG node to one of the available localities. This does two things. First, each locality then can know which expansions it needs to instantiate. Second, each locality can know to which localities a given expansion must be sent to execute the DAG.

Once the distribution of the DAG nodes is computed, each locality then instantiates the objects that are owned by that locality. Because the LCOs representing the expansions also manage the continuation of the execution after the expansion is fully computed,

shortly after their creation, DASHMM will provide a summarized form of the edges that start at the given expansion. With the expansion and out edge data prepared, the execution can begin.

4.5 DAG execution

The execution of a particular DAG comes down to the operation of the individual expansion LCO objects. Each node of the implicit DAG, which are represented with an expansion LCO, operates in the same fashion, only the specific data, and the specific out edges are different from node to node. Broadly speaking, one can divide the DAG nodes into three groups: nodes that have no input edges, nodes that have no output edges, and everything else. The are the initial, terminal, and internal DAG nodes, respectively. When execution begins, it begins in the initial DAG nodes. When they have completed their work, they activate some internal nodes, which in turn activate further nodes. Eventually, the execution comes to a terminal DAG node, representing one of the desired output quantities of the calculation. This section describes how these implicit DAG nodes operate, leading to the asynchronous, unfolding execution.

The expansion LCO participates in two kinds of actions on the underlying expansion data. The first, which will occur once for each edge ending at the DAG node, is the addition of a contribution computed elsewhere to the data of the expansion. The second, occurring only once after all the input edges to the DAG node have been satisfied, is responsible for performing the work of the edges that start at the DAG node. This latter action is also denoted the continuation action of the expansion LCO.

The effect of the contribution action on the LCO is easy to describe. The incoming expansion data is added to the expansion data represented by the LCO. The exact details of this summation is up to the kernel being implemented, but this is often just a summation of each term in the input to the equivalent term in the LCO's data.

The continuation action, which is triggered once all inputs are processed, is more involved. The purpose of the continuation action is to serve the edges that start at the current implicit DAG node. Along each edge, the data at the current DAG node is transformed, and then the contribution action is called on the target of the edge. Given that the target of an edge might be on a different locality, each edge could lead to a network message. To avoid this, DASHMM sorts the out edges by target locality, and deals with those edges together, sending a single message per locality, instead of one message per edge. Further, to keep the size of these messages small, it is the expansion data stored at the current LCO that is sent, and not the translated versions of that expansion. Once the expansion, and the list of targets and operations arrive at the remote locality, an action is performed that performs the received operations and makes the contributions to the LCOs that are the targets of the edges. These contributions, made many times from many different starting LCOs, will ultimately provide all of the inputs to other expansion LCOs, which then perform their own continuation action, leading to further evaluation, ultimately yielding the value of the potential at every target location.

To initiate this process, the initial nodes of the explicit DAG have their out edges served. This process is very similar to the continuation action for the expansion LCO, but instead of expansion data, the initial DAG nodes represent source data (such as position and charge). In the same way that multiple edges from an intermediate DAG node to the same locality are combined, multiple edges from an initial DAG node to a given locality are combined. In most cases, the edges from an initial DAG node that cross the network are all S→T edges. DASHMM places the expansion LCO for the target of a S→M edge at the same locality as the initial DAG node that is the source of the edge.

DASHMM use of asynchronous execution leads to the challenge of detecting when the evaluation is complete. To assist in this process, a second type of LCO was defined, the target LCO. This LCO has the task of tracking the number of inputs that have occurred for a given cluster of target locations. For each leaf of the target tree, there is one target LCO. These target LCOs are the representations in the implicit DAG of the terminal nodes of the explicit DAG. These LCOs act primarily as an event counter. Once each input edge has been served, the LCO triggers. To translate this into an overall detection of the completion of the evaluation, the triggering of each target LCO is connected through a continuation to another HPX-provided LCO (the and LCO). There is one such LCO for each available locality. When these are triggered, it represents that the work on that locality is finished. To detect that the entire system is finished, these one-per-locality and LCOs are connected via continuation to a single and LCO with one input per locality. This final and LCO signals that each locality is complete.

The three sets of activities, initiation, evaluation and setting up termination detection, can occur simultaneously (there are no dependencies between these actions). So DASHMM performs these three actions simultaneously. The initiation process expands out into a number of actions, each of which is responsible for initiating a subset of the initial DAG nodes. The termination detection setup leads to an action on each locality that connects the various LCOs into the web of continuations that will ultimately signal completion of the computation. And throughout, those expansion LCOs that become ready will be performing their actions. The result is a dynamically unfolding set of tasks that can occur, which get scheduled by the HPX-5 runtime system.

5 Results

To demonstrate the performance of DASHMM in distributed memory settings, a number of evaluations of multipole methods were conducted on a range of core counts to demonstrate the strong scaling of the library.

The runs reported in this section were performed on the Haswell compute nodes of Cori, a Cray XC40 supercomputer of NERSC. Each compute node has two 16-core Intel Xeon E5-2698 v3 CPUs at 2.3 GHz clock rate and 128 GB DDR4 RAM. The compute nodes are connected through the Cray Aries interconnect with Dragonfly topology.

The codes used in the test were compiled using the Intel compiler 17.0.2, with `-O3`

Table 1: The number of sources and targets for six strong scaling runs of the FMM97 method built into DASHMM.

Kernel	Distribution	N [10^6]
Laplace	Cube	100
	Sphere	90
Yukawa	Cube	100
	Sphere	60
Helmholtz	Cube	90
	Sphere	30

level of optimization. For HPX-5, the test used version 4.0.0 and was configured with Intel's TBB memory allocator and the Photon network. The test program used to collect these timings is distributed with DASHMM in the `demo/basic` subdirectory. For the test code, `DASHMMEXTRATIMING` was defined to produce the timing information that is reported here.

The source and target ensembles used in these tests contained the same number or members in each run, but the specific locations for the sources and targets were distinct. There were two distributions of sources and targets employed in these tests: in the first, the points were distributed in a unit cube; in the second, the points were distributed on the surface of a unit sphere. In each case, the charges of the source points were drawn randomly from the range [1,2] and were given a random sign.

Each test was performed for the FMM97 method for three different kernels, Laplace, Yukawa, and Helmholtz, requiring three digits of accuracy. For each combination of kernel and method, the number of points was selected by finding the largest problem that would fit in a single compute node of Cori. See Table 1 for a breakdown of problem size for each run. The length of the expansion for scaling-variant Yukawa and Helmholtz kernels depends on the depth in the tree. Therefore, the problem sizes are smaller than those for the scaling-invariant Laplace kernel. The problem size for the Helmholtz kernel is the smallest because the length of its intermediate expansion is the longest, approximately three times as long as that for the Laplace kernel.

At each scale, for each combination of kernel and distribution, the test code was executed five times, and the time for the execution phase (Section 4.5) of the calculation was recorded. The left panel of Fig. 3 shows the average of these five iterations, with curves coded by color (blue for Laplace, orange for Yukawa, and red for Helmholtz) and by marker (squares for cube data, and circles for sphere data). The speedup relative to one locality (for Cori each node has 32 physical cores) is shown in the right panel of Fig. 3.

DASHMM achieves good scaling as the resources grow by a factor of 256. A significant portion of the loss of scaling at higher locality counts is an ability of HPX-5 to accept priority hints for the tasks submitted to the system [9]. The scaling for the cube data is better for each kernel than that of sphere data because there are far fewer partition levels and far more tasks within each level for the cube data. The scaling for the Helmholtz ker-

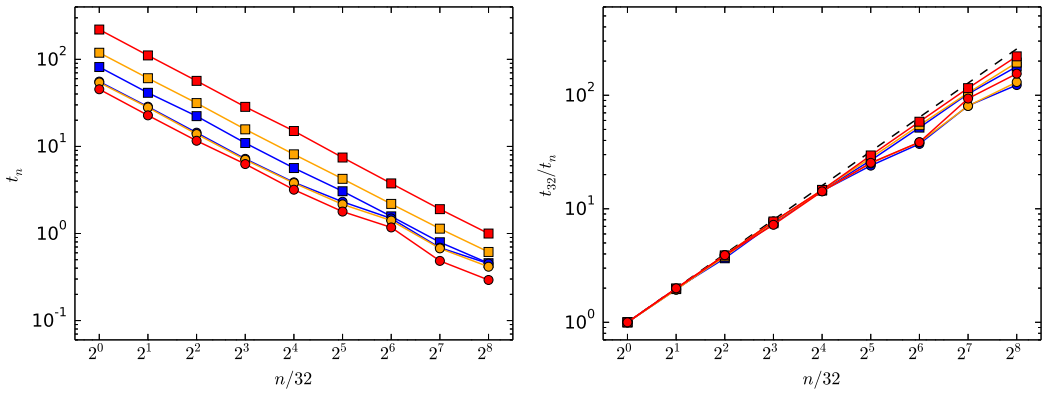

Figure 3: The execution time (left) and scaling relative to one locality (right) for the six strong scaling runs. In blue are the results for the Laplace kernel, in orange are the results for the Yukawa kernel, and in red are the results for the Helmholtz kernel. The two source and target distribution types are indicated by the marker, with square markers for cube data, and circular markers for sphere data. The horizontal axis gives the number of localities used, each of which has 32 cores ($n = 32n_l$). The largest scale run for each case used 8192 cores on 256 nodes. The data, files and production scripts for this figure are available under CC-BY [11].

nel is also uniformly better than either Laplace or Yukawa. The primary reason for this is that the expansion length for the Helmholtz kernel can be quite long, leading to a much larger grain size, even though there are fewer sources and targets for the Helmholtz case.

6 Conclusion

DASHMM is a library that provides a general framework for evaluating HMMs in parallel using an asynchronous many tasking runtime system. It provides an easy-to-use, extensible, efficient and scalable parallel evaluation with very little work on the part of application developers.

The central feature of this update is that DASHMM is now capable of working in distributed memory architectures. The resulting scaling is good, and potential improvements have been identified that can increase the scalability of the library [9]. The user of DASHMM can be oblivious to the details of the parallel execution, allowing very rapid incorporation of parallel HMMs into existing codes.

DASHMM now includes also an advanced version of FMM, called FMM97 in this paper, which used the intermediate expansion and the merge-and-shift technique. In the process of adding this capability to DASHMM, the library was also made capable of handling a wide variety of advanced methods, such as evaluating multiple interactions simultaneously. This update to DASHMM further adds two additional built-in kernels, Yukawa and low-frequency Helmholtz. After these additions, DASHMM has built-in capabilities that can serve a large number of application use cases.

The work of producing and testing this version of DASHMM has identified a num-

ber of improvements that can be made to the library. Future work on the DASHMM library will focus on priority scheduling to further improve strong scaling performance, improvements to distribution policies, and adding support for heterogeneous memory architecture, to name a few.

Acknowledgments

This work was supported by National Science Foundation grant number ACI-1440396. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U. S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] Milton Abramowitz. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*. Dover Publications, Incorporated, 1974.

[2] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, and T. Takahashi. Task-based FMM for multicore architectures. *SIAM J. Sci. Comput.*, 36:C66–C93, 2014.

[3] A. Amer, N. Maruyama, M. Pericás, K. Taura, R. Yokota, and S. Matsuoka. Fork-join and data-driven exeuction models on multi-core architectures: Case study of the FMM. *Lect. Notes. Comput. Sc.*, 7905:255–266, 2013.

[4] T. Amisaki, S. Toyoda, H. Miyagawa, and K. Kitamura. Development of hardware accelerator for molecular dynamics simulations: A computation board that calculates nonbonded interactions in cooperation with fast multipole method. *Journal of Comput. Chem.*, 24:582–592, 2003.

[5] S. S. Bindiganavale and J. L. Volakis. Guidelines for using the fast multipole method to calculate the RCS of large objects. *Microw. Opt. Techn. Let.*, 11:190–194, 1996.

[6] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM J. Sci. Stat. Comp.*, 9:669–686, 1988.

[7] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *J. Comput. Phys.*, 155:468–498, 1999.

[8] W. C. Chew, J. M. Jin, E. Michielssen, and J. M. Song. *Fast and Efficient Algorithm in Computational Electromagnetics*. Artech House, 2001.

[9] J. DeBuhr, B. Zhang, and L. D'Alessandro. Scalable hierarchical multipole methods using an asynchronous many-tasking runtime system. In *Proceedings of the 18th IEEE International Workshop on Parallel and Distributued Scientific and Engineering Computing (PDSEC 2017)*, Forthcoming.

[10] J. DeBuhr, B. Zhang, A. Tsueda, V. Tilstra-Smith, and T. Sterling. DASHMM: Dynamic Adaptive System for Hierarchical Multipole Methods. *Communications in Computational Physics*, 20:1106–1126, October 2016.

[11] Jackson DeBuhr, Bo Zhang, and Thomas Sterling. DASHMM Strong Scaling on Cori. https://figshare.com/articles/DASHMM_Strong_Scaling_on_Cori/4910519.

[12] W. Dehnen. A Hierarchical O(N) Force Calculation Algorithm. *Journal of Computational Physics*, 179:27–42, June 2002.

[13] Chirag Dekate, Matthew Anderson, Maciej Brodowicz, Hartmut Kaiser, Bryce Adelstein-Lelbach, and Thomas Sterling. Improving the scalability of parallel n-body applications with an event-driven constraint-based execution model. *International Journal of High Performance Computing Applications*, April 2012.
[14] N. Geng, A. Sullivan, and L. Carin. Fast multipole method for scattering from 3-D PEC targets situated in a half-space environment. *Microw. Opt. Techn. Let.*, 21:399–405, 1999.
[15] L. Greengard and J. Huang. A new version of the fast multipole method for screened Coulomb interactions in three dimensions. *J. Comput. Phys.*, 180:642–658, 2002.
[16] L. Greengard, J. Huang, V. Rokhlin, and S. Wandzura. Accelerating fast multipole methods for the Helmholtz equation at low frequencies. *IEEE Computational Science and Engineering*, 5:32–38, 1998.
[17] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325–348, 1987.
[18] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numer.*, 6:229–269, 1997.
[19] L. Gürel, Ö. Ergül, A. Ünal, and T. Malas. Fast and accurate analysis of large metamaterial structures using the multilevel fast multipole algorithm. *Prog. Electromagn. Res.*, 95:179–198, 2009.
[20] E. W. Hobson. *Spherical and Ellipsoidal Harmonics*. Dover, 1955.
[21] S. Jiang, Z. Liang, and J. Huang. A fast algorithm for Brownian dynamics simulation with hydrodynamic interactions. *Math. Comput.*, 82:1631–1645, 2013.
[22] H. Ltaief and R. Yokota. Data-driven execution of fast multipole methods. *CoRR*, abs/1203.0889, 2012.
[23] P. M. Morse and H. Feshbach. *Methods of Theoretical Physics*. McGraw-Hill, 1953.
[24] S. Teng. Provably good partitioning and load balancing algorithms for parallel adaptive n-body simulation. *SIAM J. Sci. Comput.*, 19:635–656, 1998.
[25] M. Warren and J. Salmon. Astrophysical n-body simulation using hierarchical tree data structures. In *SC 92': Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, 1992.
[26] M. Warren and J. Salmon. A parallel hashed oct-tree n-body algorithm. In *SC 93': Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, 1993.
[27] Michael S Warren and John K Salmon. A portable parallel particle program. *Computer Physics Communications*, 87(1-2):266–290, 1995.
[28] N. Yarvin and V. Rokhlin. Generalized Gaussian quadratures and singular value decompositions of integral operators. *SIAM J. Sci. Comput.*, 20:699–718.
[29] B. Zhang, J. Huang, N. P. Pitsianis, and X. Sun. recFMM: Recursive Parallelization of the Adaptive Fast Multipole Method for Coulomb and Screened Coulomb Interactions. *Communications in Computational Physics*, 20:534–550, 2016.