Computer Physics Communications 255 (2020) 107248

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Fast and scalable evaluation of pairwise potentials☆

S. Hughey [a,*], A. Alsnayyan [a], H.M. Aktulga [b], T. Gao [c], B. Shanker [a]

[a] Department of Elec. and Comp. Eng., Michigan State University, East Lansing, MI, USA
[b] Department of Comp. Sci., Michigan State University, East Lansing, MI, USA
[c] Department of Mech. Eng., Michigan State University, East Lansing, MI, USA

ARTICLE INFO

Article history:
Received 21 May 2019
Received in revised form 24 January 2020
Accepted 25 February 2020
Available online 2 March 2020

Keywords:
Fast multipole methods
Parallel algorithms
N-body problem

ABSTRACT

Pair potentials or kernels, $\psi(|\mathbf{r}|)$, play a critical role in a number of areas; these include biophysics, electrical engineering, fluid dynamics, diffusion physics, solid state physics, and many more. The need to evaluate these potentials rapidly for N particles gives rise to the classical N-body problem. In this paper, we present scalable parallel algorithms for evaluation of these potentials for highly non-uniform distributions. The underlying methodology for evaluating these potentials relies on the accelerated Cartesian expansion (ACE) framework that is quasi-kernel-independent with the requirement that the kernel be differentiable with known derivatives. The results presented demonstrate the accuracy control, low cost, and parallel scalability offered by this method for several example kernels and distributions of up to 5 billion particles on 16384 CPU cores. Potential applications of the algorithm include various disciplines of computational physics, engineering, machine learning, among others.

1. Introduction and problem statement

The evaluation of pairwise interactions between N particles in $\mathbb{R}^d$, $d \in \mathbb{Z}^+$, commonly referred to as the N-body problem, is a classic problem in computational science with a wide variety of applications. Generally, the problem may be defined as the evaluation of the sum

$$\Phi(\mathbf{r}_i) = \sum_{j=1}^{N} \psi(\mathbf{r}_i, \mathbf{r}_j)\sigma_j, \quad i = 1, \ldots, N \tag{1}$$

where $\psi(\mathbf{r}, \mathbf{r}') : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C}$ is called the kernel function, $\Phi(\mathbf{r})$ is the potential at the point $\mathbf{r}$, and σ_j is the coefficient of the jth source particle located at $\mathbf{r}_j \in \mathbb{R}^d$. Applications of this equation range from mathematical physics wherein the kernel is a Green's function used to compute fields from sources or modern machine learning/interpolation methods. Example kernel functions that are often used are given in Table 1. Several machine learning techniques, increasingly popular in recent years, and interpolation methods [1–5] also rely on the N-body calculation. As is well known, the principal difficulty with the N-body calculation is that it is an inherently dense computation, resulting in $\mathcal{O}(N^2)$ computational complexity for evaluation of (1). This operation may also be interpreted as a matrix–vector product $Ax = b$, with the kernel matrix given by $A_{ij} = \psi(\mathbf{r}_i, \mathbf{r}_j)$ and vectors $x_j = \sigma_j$, $b_i = \Phi(\mathbf{r}_i)$. As N becomes very large the cost of the matrix–vector product becomes prohibitive, and fast, error-controllable evaluation methods become necessary.

Given the vast applicability of the N-body calculation, a number of researchers have contributed to a robust and varied literature on fast algorithms for this problem. Perhaps the most impactful of these are the fast multipole methods (FMMs), first introduced by Greengard and Rokhlin for the Laplace potential [6,7]. In these methods, elements of the kernel matrix are not explicitly formed; instead, the FMM is a method for rapidly computing the application of the kernel matrix to a vector. One should also mention the Barnes–Hut (or treecode) algorithms [8], upon which the fundamental ideas of the FMM were based. For a discussion of modern FMMs, it is useful to define a coarse taxonomy of methods. Kernel functions of interest typically fall into one of two categories: oscillatory (kernels which encode phase information, e.g., Helmholtz potential) and non-oscillatory (kernels which do not encode phase, e.g., Laplace potential). Designing an FMM for oscillatory kernels requires particular care which will not be discussed here; the interested reader is instead referred to Refs. [9–13]. FMMs may additionally be categorized as either kernel-dependent FMMs, which are based on a kernel-dependent factorization of a particular kernel, or kernel-independent FMMs, which are not. When applicable, kernel-dependent FMMs may be considerably more efficient than their kernel-independent counterparts thanks to the *optimal* exploitation of particular properties of the factorization; see, for instance, the Ref. [9]. However, implementation of kernel-dependent FMMs is often considerably more involved than that of kernel-independent FMMs.

☆ The review of this paper was arranged by Prof. David W.Walker.

* Correspondence to: 428 S. Shaw Ln, E. Lansing, MI 48825, United States.
E-mail address: hugheyst@msu.edu (S. Hughey).

https://doi.org/10.1016/j.cpc.2020.107248
0010-4655/

Table 1
Some examples of common kernel functions.

Potential function	$\Phi(r)$
Laplace	$1/r$
Yukawa	$e^{-\gamma r}/r$
Lattice Gas	$\log(r)$
Helmholtz	$e^{-ik_0 r}/r$
Retarded potential	$\delta(t-r/c)/r$
Free-particle Schrödinger	$Ae^{iBr^2/2t}/t^{3/2}$
Gaussian	$e^{-ar^2/2\sigma^2}$
Polyharmonic spline	$r^k \log r$
Multiquadric	$\sqrt{1+\alpha r^2}$

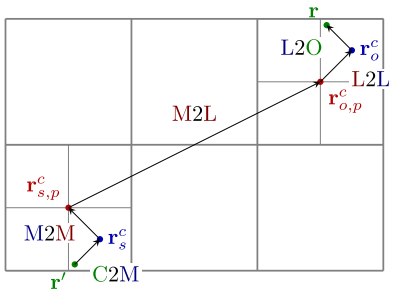


Fig. 1. Illustration of the operations in the ACE algorithm and notation used in this section.

Within the class of kernel-independent FMMs, there are several options, each with their own strengths and weaknesses. As each of these methods can in some sense be viewed as a local polynomial approximation, we shall denote the order of approximation as P. The KIFMM algorithm [14] offers acceleration with low $\mathcal{O}(NP^2)$ computational complexity thanks to a framework of surface integrals radiating equivalent source densities, requiring only a surface discretization of each cell. However, the use of the integral radiation operator restricts the applicability of this algorithm to kernels satisfying a radiation condition, e.g. $\mathcal{O}(r^{-\nu})$, $\nu \geq 1$. The black-box FMM (bbFMM) [15] employs tensor-product Chebyshev interpolation, requiring a volumetric discretization of each cell which implies an $\mathcal{O}(NP^3)$ cost due to dense matrix operations between cells with $(P+1)^3$ interpolation nodes each. However, implementation is straightforward and the algorithm can be used with any smooth kernel. Additionally, the cost can be reduced via low-rank factorization or FFT acceleration [11,16].

Another method developed in 2007, is the accelerated Cartesian expansion (ACE) method [17] that uses the Taylor series expansion in Cartesian tensor form to systematically express addition theorems for *any* arbitrary smooth kernel. More importantly, ACE does not require kernel functions *to satisfy the radiation condition.* The structure of the method offers numerous computational advantages elucidated and demonstrated in Refs. [17–20]. Further, while its polynomial representation implies a cost of $\mathcal{O}(NP^3)$, further analysis reveals the constant is small and the number of degrees of freedom per cell is asymptotically about 1/6 of that of the bbFMM, rendering the algorithm highly efficient for low P. In some sense, ACE straddles the boundary between kernel-dependent and kernel-independent: While the KIFMM and bbFMM require only a function handle to evaluate the kernel, ACE requires knowledge of the kernel's derivatives. Fortunately, for many of the smooth kernels for which FMM-like methods are desired, these are often available analytically.

Parallelization of the aforementioned algorithms has also been the subject of intense focus over recent years. The advent of powerful supercomputers with massive numbers of CPU cores has driven interest in extremely large N-body calculations involving millions to billions of particles. Special-purpose algorithms for the Laplace kernel have been perhaps the most well-studied in this regard [21,22], but several parallel algorithms for kernel-independent FMMs have also been put forth and have exhibited similar scaling properties [14,23,24]. The sheer size of the problems of interest and variations in the density of the underlying particle distribution present challenges in parallelization.

This paper presents a scalable parallel algorithm for evaluating N-body sums involving arbitrary non-oscillatory potentials, including those that do not decay as $r \to \infty$, via the ACE algorithm. Building upon [18], our contributions are:

- Algorithms for efficiently constructing and load-balancing highly non-uniform trees;
- Controllably-accurate ACE operators for non-uniform trees;
- Optimizations to the matrix–vector product;
- Performance, accuracy, and scalability for non-decaying kernels over large and highly non-uniform distributions;
- Direct performance comparison of our method with a highly -optimized KIFMM implementation.

The paper is organized as follows. Section 2 gives a brief overview of the ACE algorithm and establishes notation. Section 3 describes the new parallel algorithm, and Section 4 gives numerical and performance results demonstrating the accuracy and efficiency of the presented approach, and a comparison of its performance against the KIFMM. Appendix A provides insight how efficiencies can be realized in applying ACE to evaluate Stokes potentials. Finally, Appendix B provides the theorems necessary to completely define ACE (for completeness purposes).

2. Mathematical preliminaries

In this section, we give a succinct summary of the relevant aspects of the ACE algorithm and highlight some useful features. For more details, the reader is referred to Ref. [17]. Consider the N-body problem described in (1). Let $\Omega \in \mathbb{R}^3$ denote an axis-aligned cubical domain containing all of the particles. Like all FMM-style algorithms, ACE relies on a hierarchical octree decomposition of the space Ω, and the number of levels in the tree will be denoted by L, where level 1 represents the root of the tree and level L is the leaf level, or the finest level of refinement.

Consider two boxes Ω_s and Ω_o that contain source and observers, respectively, in the oct-tree decomposition of Ω, and let the parents of Ω_s and Ω_o be denoted using Ω_s^p and Ω_o^p. Centers of these boxes are denoted using $\mathbf{r}_s^c$, $\mathbf{r}_o^c$, $\mathbf{r}_{s,p}^c$, and $\mathbf{r}_{o,p}^c$. To set the stage, we start with some remarks and notations; (i) an nth rank totally symmetric tensor $\mathbf{A}^{(n)}$ contains $(n+1)(n+2)/2$ independent components as opposed to 3^n components; (ii) in this compressed form this tensor can be represented using $A^{(n)}(n_1,n_2,n_3)$ where $n = n_1+n_2+n_3$; (iii) an example of such a tensor is the polyadic associated with $\mathbf{r}$ which is given by $\underbrace{\mathbf{r}\mathbf{r}\cdots\mathbf{r}}_{n \text{ times}} = \mathbf{r}^n$ and can be represented in compressed form as $r^{(n)} = x^{n_1}y^{n_2}z^{n_3}$; (iv) an m-fold contraction between two tensors $\mathbf{A}^{(n)}$ and $\mathbf{B}^{(m)}$ is denoted using $\mathbf{A}^{(n)} \cdot m \cdot \mathbf{B}^{(m)} = \mathbf{C}^{(n-m)}$; and (v) a direct product between two tensors can be written as $\mathbf{C}^{(n+m)} = \mathbf{A}^{(n)}\mathbf{B}^{(m)}$.

The heart of ACE is the Taylor series expansion, which provides a natural framework for developing addition theorems. Consider a source located at a point $\mathbf{r}'$ observed at a point $\mathbf{r}$. For convenience we adopt the notation $\mathbf{r}-\mathbf{r}' = \mathbf{X}+\mathbf{d}$, where $\mathbf{X} = \mathbf{r}_o^c - \mathbf{r}_s^c$ and $\mathbf{d} = (\mathbf{r}-\mathbf{r}_o^c) + (\mathbf{r}_s^c - \mathbf{r}')$; see Fig. 1 for reference. A Taylor series expansion of the kernel function $\psi(\mathbf{X}+\mathbf{d})$,

$$\psi(\mathbf{X}+\mathbf{d}) = \sum_{n=0}^{\infty} \frac{1}{n!}\mathbf{d}^n \cdot n \cdot \nabla^n \psi(\mathbf{X}) \tag{2}$$

for $|\mathbf{X}| > |\mathbf{d}|$, can express an addition theorem for any sufficiently smooth, non-oscillatory kernel function. In practice, this sum is truncated at some finite P, with increasing accuracy for larger values of P. From this representation, the N-body sum (1) may be recast as

$$\Phi(\mathbf{r}_i) = \Phi_N(\mathbf{r}_i) + \Phi_F(\mathbf{r}_i), \tag{3}$$

where $\Phi_N(\mathbf{r}_i)$ represents contributions to $\Phi(\mathbf{r}_i)$ from particles in octree boxes adjacent to that containing the ith observation point $\mathbf{r}_i$. The remaining (far) contributions, given by $\Phi_F(\mathbf{r}_i)$, are calculated using the addition theorem.

The ACE algorithm follows the standard fast multipole procedure; for particulars, the reader is referred to [17]. First, multipole expansions of sources are computed for each leaf-level box and aggregated up the octree. Next, local expansions are formed for each octree box by accumulating translated multipole expansions in its interaction list. As implied by (2), a local expansion may be interpreted as the coefficients of the Taylor polynomial representing the influence of all particles exterior to the near-field of its octree box. Finally, local expansions are disaggregated in a top-down fashion to the leaf level, where they are used to compute the potential at each observer particle. For non-uniform particle distributions it is advantageous to use an adaptive octree which can conform to the local distribution density (see Section 3). Two additional steps involving interactions between octree boxes of different sizes are required [13,25]. Because the condition $|\mathbf{d}| < |\mathbf{X}|$ is not satisfied in three dimensions for these interactions, the operators involved take the form of either (i) directly evaluating multipole expansions at observer particles or (ii) summing the influence of individual source particles onto local expansions; they are trivially derived by subsuming either the observer or source parts, respectively, of $\mathbf{d}$ into the vector $\mathbf{X}$, yielding a valid addition theorem.

We now state several interesting and/or useful properties of the algorithm. First, it has been shown that the operators involved in aggregation and disaggregation are *exact*, i.e. they incur no numerical error beyond the specification of P and these are *independent of the height of the tree*. Second, the source-side multipole expansions do not involve the kernel function; hence, multiple potentials can be computed at once from the same source data, provided one can store a set of local expansions for each kernel considered. Third, operators for going up and down the tree (between any two consecutive levels) differ only by a constant, and are easily precomputed. The same is true for certain kernel functions. Fourth, the k-fold gradient of the potential is trivially evaluated using the formula [17]

$$\nabla^k \Phi(\mathbf{X} + \mathbf{d}) = \sum_{n=0}^{P} \frac{1}{(n-k)!} \mathbf{d}^{(n-k)} \cdot (n-k) \cdot \nabla^n \psi(\mathbf{X}). \tag{4}$$

As we will demonstrate, this property is especially useful for evaluating multiple distinct potentials which derive from a common kernel function, or in some cases reduces the overall cost when a potential can be expressed in terms of differential operators acting upon a simple kernel function (like the Stokes potential). Pertinent details for application to the Stokes potential are presented in Appendix A.

3. Parallel ACE algorithm

In this section, we outline our parallel algorithms for constructing, adapting, and load balancing the tree, as well as parallel potential evaluation. While parallelization of FMMs is well-documented in the literature [3,23,26], as we will demonstrate in Section 4, redundant M2L computations on processes sharing the same local expansions in the commonly used locally essential tree (LET) based algorithms would significantly hamper the performance of our proposed framework, particularly for high-accuracy calculations. The M2L stage incurs the largest computational cost in ACE in terms of P, whereas in methods such as KIFMM this is not the case [23]. For this reason, our parallel evaluation strategy differs from the usual LET based implementations, and therefore we present our parallel implementation in detail.

3.1. Construction of the distributed octree

Construction of the distributed octree starts by partitioning the input particles equally among all processes. On each process, the equi-distributed particles are first hashed into Morton keys, which is a binary encoding of the leaf boxes according to their space-filling Morton-Z curve ordering. A parallel bucket sort is then used to assign each process a distinct set of leaves contiguous in the Morton-Z ordering such that each process gets a roughly equal number of particles. This initial partitioning is performed at the finest refinement level, *i.e.*, at level L, in preparation for the ensuing adaptive tree coarsening and load balancing stages, and is concluded with a local post-order traversal tree construction on each process from leaves all the way up to the root.

The above partitioning scheme necessarily incurs duplicate copies of internal tree nodes, corresponding to common ancestors of leaf nodes residing on different processes. Such nodes are referred to as *plural* nodes. Treatment of computations associated with plural nodes is one aspect of our parallelization scheme that differentiates it from the LET based implementations. Hence, we provide some terminology to aid the discussion. For any plural node, the process with the highest rank which owns a copy is designated as the *resident* process, and as such is deemed responsible for all its tree-based interactions. Other processes with a copy of the plural node are called *users* of the node, and are directed only to send and receive data to and from the resident process. A plural node is referred to as a *shared* node on its resident process, and the users' copies are called *duplicate* nodes. As a consequence of the post-order traversal tree construction, it can be shown that all duplicate nodes in a process' local sub-tree appear consecutively at the end of the post-order traversal sequence. Shared nodes may appear anywhere within the post-order sequence, though they tend to appear toward the beginning. As we will discuss, these properties allow effective overlapping of communication and computation in the potential evaluation stage.

3.2. Adaptive tree

Random (or homogeneous) distributions of source or observer particles are well-represented by a uniform tree structure. Within such distributions, the number of particles per box is approximately constant across the entire simulation domain, facilitating the linear scaling of the ACE algorithm or any other FMM implementation for that matter. For non-uniform distributions though, *i.e.*, those with sub-regions of relatively high particle density, the minimum box size must be decreased to prevent near-field costs from dominating the computation. Hence, in a uniform tree framework, the densest region of discretization dictates the leaf box size throughout the *entire* tree. The result is large swaths of space with a poor work-to-particle ratio in sparsely populated regions, degrading the cost scaling of parallel ACE computations.

To prevent such inefficiencies, we adapt the tree structure to the particle distribution so as to restore the approximate uniformity in leaf box population throughout the tree. Adaptation of trees for N-body simulation has been well-studied and has become a standard feature of modern tree-based simulation

methods for non-oscillatory kernels [24,27–29]. We employ a bottom-up scheme for both constructing and merging the tree to minimize communication costs. Our parallel algorithm for merging the distributed octree is given in Algorithm 1. Simply put, this algorithm starts with a uniform tree constructed according to the description in Section 3.1 and merges each set of sibling leaf boxes into their parents, if the total number of particles in these leaf boxes do not exceed a pre-determined particle threshold M. This scheme ensures that leaf boxes in more sparsely-populated regions contain roughly as many points as those in the dense regions, thereby avoiding excessive tree interactions (in sparse regions) or expensive near field computations (in dense regions).

Algorithm 1 Parallel algorithm for merging the tree with the option to impose 2:1 balance constraint

Define:
$\mathcal{T}_{loc}$: local subtree (including plural nodes)
$\mathcal{T}_{loc}^{\ell}$: local subtree at level ℓ
M: maximum number of points per leaf box
$S(b)$: set of siblings of a box b
$n(b)$: number of points contained within the *complete* subtree rooted at box b
$\mathcal{N}(b)$: set of box b's near-neighbors at the same level
Votes: "1": vote to merge; "0": indeterminate; "−1": veto
`Votes` $\leftarrow 0$
for $\ell = L, L-1, \ldots, 4$ **do**
  $\mathcal{C} \leftarrow \emptyset$
  for each $b \in \mathcal{T}_{loc}^{\ell}$ **do**
    if `Votes`(b)==0 **then**
      if $n(P(b)) \leq M$ **then**
        `Votes`$(S(b)) \leftarrow 1$
      else
        `Votes`$(S(b)) \leftarrow -1$
      end if
    end if
    if `Votes`$(b) \leq 0$ **and** 2:1 balance desired **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{N}(P(b))$
    end if
  end for
  `Votes`$(S(\mathcal{C})) \leftarrow -1$ {Global step; involves communication of vetoes to resident processes of boxes in $\mathcal{C}$}
  Synchronize votes of level ℓ plural nodes and their siblings; vetoes dominate
end for
Prune each $b \in \mathcal{T}_{loc}$ and re-assign particles accordingly

3.2.1. Evaluation of ACE interactions in adaptive trees

In adaptive octree algorithms, it is necessary to evaluate interactions between boxes of different sizes. Fig. 2(a) gives an illustrative example. As usual, these interactions are classified into U, V, W, X-lists based on the adjacency and relative size of source and observer box pairs [7,25]. For the sake of completeness, these lists are defined as follows:

- **U-list**: source and observer boxes are adjacent and are both leaves, *i.e.*, they are within the near field of each other;
- **V-list**: source and observer boxes are not adjacent but their parents are, *i.e.*, they are inside the far field of each other;
- **X-list**: source and observer boxes are not adjacent, but the observer box is (i) adjacent to the parent of the source box, (ii) a leaf, and (iii) larger than the source box, corresponding to a cross-level interaction with an observer node higher up in the octree;
- **W-list**: reciprocal interactions of those in the X-list, corresponding to a cross-level interaction with an observer node lower down in the octree.

3.2.2. 2:1 balance constraint

For highly non-uniform distributions, unconstrained merging of the tree results in a tree with a matching degree of non-uniformity. While this is normally desirable from the perspective of reducing computational costs (less tree nodes means less number of interactions overall), it has a downside in terms of memory utilization. In a uniform tree where no X-list or W-list interactions exist, there can be at most 6^3 (total number of boxes in the neighborhood of a box) - 3^3 (number of near field boxes) $= 189$ boxes in the far field of a box. Leveraging symmetries and scale invariance of certain kernels can significantly reduce the number of translation operators needed for V-list interactions. However, in regions containing sharp discontinuities in leaf box size, the number of X- and W-list interactions can become quite large, and the number of unique translation operators needed to perform the corresponding X- and W-list interactions can also be very large. The *2:1 balance constraint* [24,29] remedies this storage problem by disallowing adjacent leaf boxes to differ in size by more than a factor of two, significantly reducing the number of different X and W interactions and the memory overhead for storing their particle-specific translation operators. Algorithm 1 for merging the tree includes the option to impose this constraint.

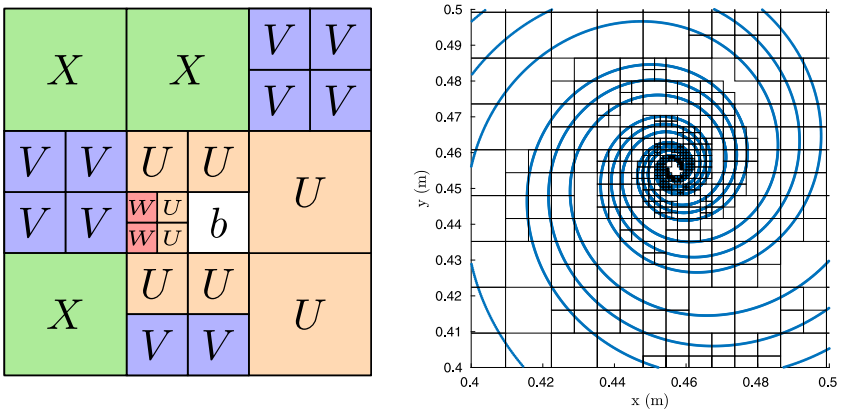
3.3. Load balancing

After the tree merging procedure under the 2:1 balance constraints is complete, each process is left with a subset of the original uniform tree. As such, the computational profile is sufficiently different from the original tree that re-balancing the load on processes is necessary. To accomplish this, we follow an empirical load balancing strategy where we estimate the work attendant to each node in the distributed tree and aim to assign them in a load-balanced manner.

The cost per leaf is determined as follows. As part of the initialization operations, we first time a set of dummy operations to obtain cost estimates for each of the U, V, W, and X-list interactions. The cost for each node in the tree is then determined by multiplying the number of each interaction by its corresponding cost estimate, and summing these costs across all interaction types. As with the initial partitioning, we partition the distributed tree by determining a set of separators between contiguous chunks of leaf boxes. Therefore we account for the costs of the interior nodes by percolating their estimated costs down to the leaf boxes and adding them to the extant cost estimate at each leaf. A variant of Algorithm 1 from [29] is then used to determine $P-1$ locations at which to split the leaf level Morton curve for a re-balancing of the computational load. While the overall strategy is similar to the costzones approach of [27,28], our strategy accounts for the work of the entire tree instead of just the leaves. This is important for surface geometries with non-uniform distributions, as the amount of work above the leaves is not negligible.

3.4. Evaluation of the potential in parallel

The parallel potential evaluation is performed in three stages: The upward pass (M2M), translation (M2L), and the downward pass (L2L). Algorithm 2 describes the M2M (upward pass) stage of our implementation, which essentially entails shifting the multipole data of each tree node to the center of its parent box and aggregating the shifted multipole data from all siblings. Each process starts processing the nodes in its local subtree from right-to-left in the post-order traversal sequence. By choosing to go from right-to-left, we ensure that duplicate nodes are encountered toward the beginning of the M2M stage, and shared nodes



(a) (b)

Fig. 2. (a) Graphical illustration of the interaction lists in the adaptive tree for a leaf box b; (b) example of non-uniform distribution with non-uniform tree in two dimensions.

are encountered toward the end of this stage. Using MPI's non-blocking `Ireduce` collective calls, this pattern facilitates overlapping the communication operations for plural nodes whose children, by definition, reside on different processes with M2M computations of other nodes.

The translation stage commences on each process once the local upward pass is complete. This stage includes three substages for computations of X, V and W lists. First, X-list interactions are handled by sending the required multipole expansions to processes with X-list observers, where the observed potentials are computed. Next, V-list interactions are carried out. Typically, this is the most expensive stage of any fast multipole-like method. To hide communication overheads, the source multipole expansions to be exchanged are divided up into packets and communicated to processes that need this information. As each source expansion is received in full, all its V-list interactions are computed, making good use of temporal locality. Once all source expansions in the current packet are exhausted of work, the next packet is constructed and communicated using non-blocking primitives to facilitate overlapping of communication and computation. This process continues until all remote V-list interactions are completed. Local V-list interactions, *i.e.*, those in which both the source and observer boxes belong to the same process, are computed next. Finally, W-list interactions are computed by exchanging the source weights, i.e. u_i in (B.1), and applying the appropriate translation operators to form the local expansions of these sources.

To complete the evaluation, Algorithm 3 describes the L2L (downward pass) stage which involves re-centering and adding parents' local expansions to their childrens'. Contrary to the M2M stage, here we employ a pre-order traversal of the local subtree so that the shared nodes are now encountered at the beginning and the duplicates are encountered toward the end of the L2L stage. This way, by using MPI's non-blocking `IBcast` primitive, we ensure that broadcast for a shared node can be completed in the background, while L2L computations of other nodes are being performed. Also, the result of the V-list interaction evaluations for duplicate nodes is communicated to users during the downward pass, again using non-blocking primitives for overlapping communications with computations.

As described above, updating multipole and local expansions for nodes that are shared between processes are performed efficiently in our implementation using asynchronous communications. We note that at any level, the local subtree of any process can have at *most* two plural nodes — one shared, one duplicate. It follows, then, that we may create a total of $2L$ MPI communicators, each with groupings of shared nodes and their users, so that we can take advantage of non-blocking variants of MPI's reduce and broadcast operations.

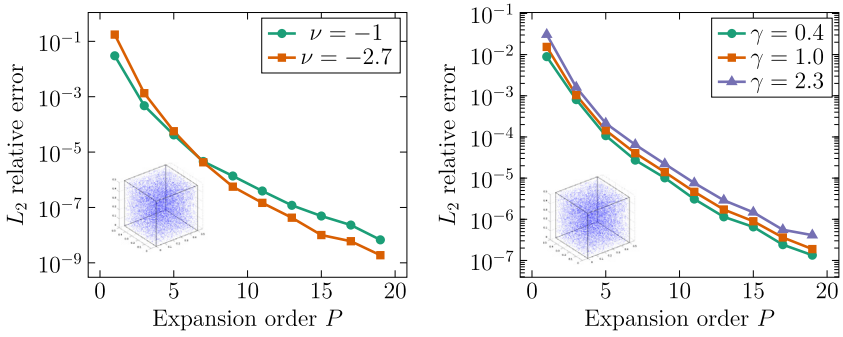
Algorithm 2 Local multipole-to-multipole (M2M) computation with interleaved update of plural nodes

1: Define `pid` as the process rank
2: Define $M(b)$ as the multipole expansion for node b
3: Define $T(b)$ as a temporary storage space of the same size as $M(b)$ unique to node b; only required for plural nodes
4: Define Reqs$(\cdot)$ as the array of request handles for asynchronous comms.
5: $r \leftarrow 0$ {r is the request counter}
6: $\mathcal{R} \leftarrow \emptyset$ {Set of update nodes}
7: **for** each $b \in \mathcal{T}_{loc}$ in right-to-left post-order **do**
8: **if** b is a leaf **then**
9: Calculate $M(b)$ from particles of b
10: **else**
11: **for** each child c of b **do**
12: $M(b) \leftarrow M(b) + \texttt{M2M}(c, b)$
13: **end for**
14: **if** b is a plural node **then**
15: $r \leftarrow r + 1$
16: `MPI_Ireduce`($M(b)$,$T(b)$,Reqs(r),Root(b),`MPI_SUM`)
17: **if** `pid` $==$ Root(b) **then**
18: $\mathcal{R} \leftarrow \mathcal{R} \cup b$
19: **end if**
20: **end if**
21: **end if**
22: **end for**
23: `MPI_Waitall`(r,Reqs($1 : r$))
24: **for** each $b \in \mathcal{R}$ **do**
25: $M(b) \leftarrow T(b)$
26: **end for**

We note that our evaluation algorithm differs from that of the common LET-based implementations in that we avoid duplicating V-list interactions, which are typically the most computationally expensive part of any fast multipole-like algorithm (see Section 4). Each V-list interaction is computed exactly once, and the resident process is responsible for such computations.

4. Results

In this section, we present an array of results demonstrating error convergence and performance of the presented algorithm and its parallel implementation.



(a) $r^{-\nu}$ potential (b) Yukawa potential $e^{-\gamma r}/r, \gamma > 0$

Fig. 3. Far-field only error convergence vs. ACE expansion order for different kernel functions.

Algorithm 3 Local-to-local (L2L) computation with asynchronous update of plural nodes

```
Define L(b) as the local expansion for node b
r ← 0
for each b ∈ T in pre-order do
  if b is a leaf then
    Calculate potentials due to L(b) for each particle in b
  else
    if b is a plural node then
      r ← r + 1
      MPI_Ibcast(L(b),Reqs(r),Root(b))
      if b is a duplicate node then
        MPI_Wait(Reqs(r))
      end if
    end if
    for each child c of b do
      if c is not a duplicate node then
        L(c) ← L(c) + L2L(c, b)
      end if
    end for
  end if
end for
```

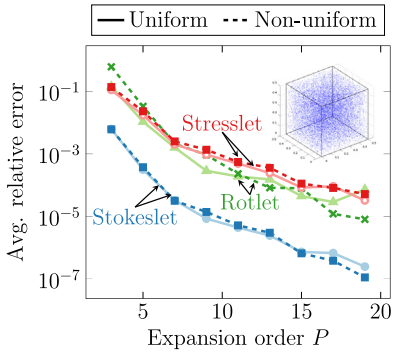


Fig. 4. Convergence of the Stokes potentials for uniform and non-uniform volume distributions.

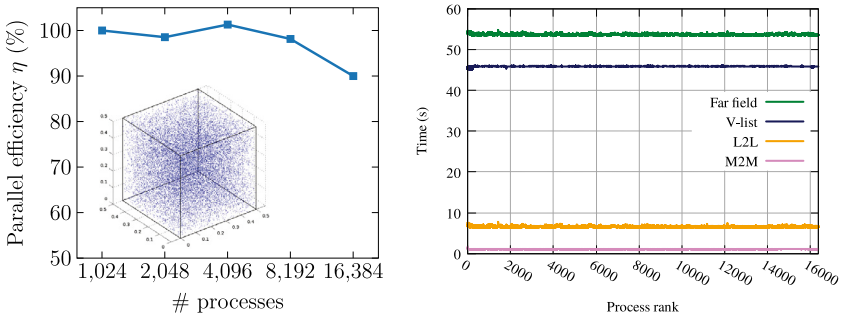
4.1. Error convergence

We first examine the error convergence of the ACE algorithm applied to the $r^{-\nu}$, Yukawa, and Stokes potentials (the Stokeslet, rotlet, and stresslet; see the Appendix for further details). The first two are computed using scalar sources, while the Stokes potentials are calculated using vector-valued sources. The error for Stokes potentials is compared to the analytical by taking an inner product of the tensor-valued potential with each observer's polarization vector. We first evaluate the $r^{-\nu}$ and Yukawa potentials for 3.125 million points within a 0.64 m cube. The distribution is mapped onto a tree with a leaf box diameter of $d_0 = 0.01$ m, yielding a 7-level tree with 12 points per box on average. A single buffer box is used for the far-field. Fig. 3 shows convergence in the L_2 error for the far-field with increasing expansion order $P = 1, 3, \ldots, 19$ for both kernels. Different parameters are used to control the growth or decay of the kernels as r increases.

We next consider the evaluation of error in Stokes potentials for a collection of points within a cube with both uniform and non-uniform spatial distributions. Both distributions are characterized by 3.125 million points within a 0.64 m cube. In the case of the uniform distribution, we employ a uniform tree with increasing box size as P is increased to minimize runtime. The non-uniform distribution is generated by generating random points within the unit cube and raising the generated x, y, z positions to the powers of 1.2, 0.7, and 1.7, respectively, before rescaling the results to fit into the 0.64 m cube. In this case, we use a non-uniform tree with 2:1 balance and increase the minimum box size with P to approximately minimize runtime according to the uniform-tree cost estimate. The error convergence with increasing P for both distributions is shown in Fig. 4. The error metric used here relies on randomly sampling the analytical fields at one observer on each process, ensuring a good spatial distribution of observers, and comparing it with the computed potential using ACE. For each selected particle, the error in the potential relative to the analytic solution is computed, and the average of these errors is reported.

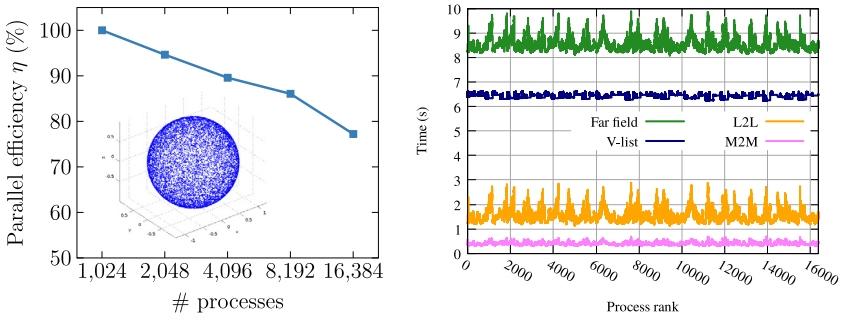
4.2. Parallel performance

Finally, we examine the performance of the algorithm in evaluating all three Stokes potentials simultaneously from the kernel $\psi(\mathbf{r}) = r$ using the formulas given in Appendix A. We empirically selected the box sizes for best performance. These results were obtained on the Haswell partition of the Cori supercomputer at the National Energy Research Scientific Computing Center (NERSC). This cluster comprises 2388 compute nodes with two sockets each, populated by 16-core Intel Xeon E5-2698 v3 "Haswell" CPUs running at 2.3 GHz and 64 GB DDR4 RAM at 2133 MHz per socket. The algorithm was implemented in Fortran 90 using double-precision arithmetic and parallelized strictly in distributed-memory fashion using MPI. The code was compiled using the Intel compiler version 18 with optimization and architecture-specific instructions using the `-O3 -xHost` flags. For these runs we use vector-valued sources in $\mathbb{R}^3$ and evaluate the far-field only, selecting $P = 7$ to give $\mathcal{O}(10^{-5})$ accuracy for the Stokeslet and $\mathcal{O}(10^{-3})$ accuracy for the rotlet and stresslet.



(a) Strong scaling (b) Per-process timings

Fig. 5. Strong scaling and per-process timings for Stokes kernel evaluation on a uniformly-distributed volume geometry with 5 billion particles.



(a) Strong scaling (b) Per-process timings

Fig. 6. Strong scaling and per-process timings for Stokes kernel evaluation on a uniform spherical distribution of 1.024 billion points.

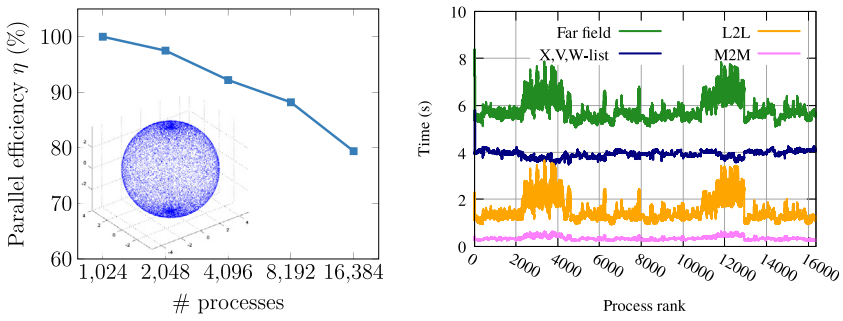
We next consider a uniform random distribution of 5 billion randomly-oriented vector sources with unit norm inside a cube of diameter 8 m. The minimum box size was set to 0.015 m, yielding an 11-level uniform tree with 33 particles per leaf box on average. Fig. 5(a) shows the parallel efficiency (strong scaling) of this distribution up to 16,384 processes with respect to 1024 processes. The increase in efficiency from 2048 to 4096 processes is due to the fact that the distribution is randomly re-generated for each run, given the extremely large number of particles. The efficiency is over 90% for all cases. Fig. 5(b) shows the timings for each process for the 16,384 process case. The C2M stage requires on average 0.54 s while the L2O stage, modified for the Stokes potentials, requires 6.05 s, about an 11.2X increase. Most of this increase is due to the calculation of the stresslet potential. The overall computational time is dominated by the V-list calculation, taking over 45 s of the 54 s, underscoring the importance of avoiding redundant V-list calculations related to duplicate nodes at upper levels of the tree.

We now consider uniformly and non-uniformly distributed sources on the surface of a sphere. Both distributions comprise 1.024 billion particles. For the uniform distribution, the sphere diameter is 2 m. A uniform tree is used with a leaf box size of 5×10^{-4} m, yielding a 13-level tree with leaves containing 15 particles per box on average. The strong scaling and per-process timings on 16,384 processes for the uniform distribution are shown in Fig. 6. A mock distribution is also shown inset in the strong scaling plot. In this case, the efficiency is over 77%. The choppiness of the L2L stage timings is due to the fact that the number of particles assigned to each process is uneven; instead, the load balancing algorithm aims to balance the overall computational load. As the bulk of the work lies in the V-list evaluation, Fig. 6 suggests the algorithm does a reasonably good job.

In the case of the non-uniform distribution, the sphere has a diameter of 8 m, and the particles are clustered around the north and south ($\pm z$) poles (as depicted inset in Fig. 7(a)). The minimum box size is chosen as 6.25×10^{-7} m in diameter, resulting in a 25-level tree. We set $s_{max} = 50$ particles per box, resulting in about 18 particles per box on average and distributing leaves over the bottom 16 levels of the tree. Despite the extremely non-uniform distribution of particles, the strong scaling shown in Fig. 7(a) is still as high as 78% efficient on 16,384 processes. In addition, the per-process timings presented in Fig. 7(b) for the far-field evaluation suggest that the load balancing algorithm does its job reasonably well. Again, the choppiness here is due to the fact that particles are not evenly distributed across processes, and processes with high particle counts spend considerable time in the L2O stage. Those with the most particles stick out in these plots.

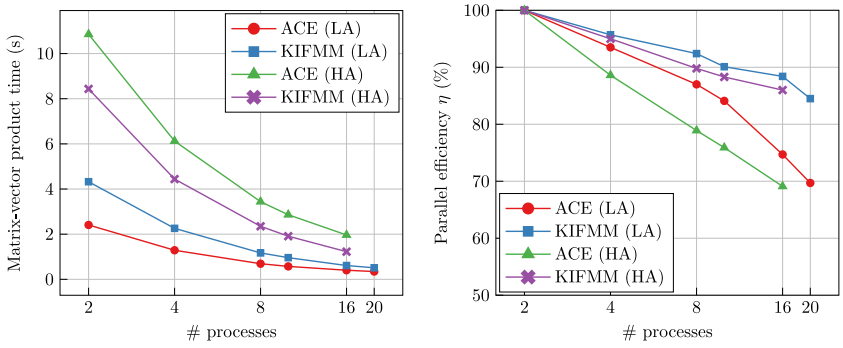
4.3. Performance comparison with KIFMM

In this section, we compare the performance of our parallel algorithm with that of the point-based kernel-independent FMM [14,23] implemented in the open-source PVFMM package [24]. We selected this code as a benchmark because it is perhaps the most well-known code of its kind. Its authors have extensively optimized the algorithm and implementation, and they have compared it with other contemporary codes [24,30,31]. The runs in this section were performed on a single compute node with an Intel (R) Xeon (R) Gold 6148 CPU at 2.40 GHz with 83 GB RAM. Because our implementation does not make use of multithreading, we considered strictly MPI parallelism. We did not do any detailed manual optimization of our implementation of the parallel ACE algorithm, instead simply relied on the `-O3 -xHost -ipo` compiler optimization flags for the Intel Fortran compiler version 19.0.3.199 and Intel MPI 2019 Update 3 for Linux. The geometry used in this experiment is a volumetric uniform distribution of $N = 500{,}000$ scalar points within a cube with side length 1 m. The kernel function is the Yukawa kernel



(a) Strong scaling (b) Per-process timings

Fig. 7. Strong scaling and per-process timings for Stokes kernel evaluation on a non-uniform spherical distribution of 1.024 billion points.



(a) Timings (b) Strong scaling

Fig. 8. Timings and strong scaling behavior of a matrix–vector product for ACE and KIFMM in lower-accuracy (LA) and higher-accuracy (HA) modes.

$\psi(|\mathbf{r}|) = \frac{e^{-\gamma r}}{r}$ with $\gamma = 1$. We ran both codes at two target accuracy settings (in the sense of the L_2 relative error), first at the lower accuracy of $\varepsilon = 10^{-4}$ and then at the higher accuracy of $\varepsilon = 5 \times 10^{-7}$. In each case, the number of points per leaf box was tuned to the best of our ability to optimize the execution time for both codes. Table 2 summarizes the simulation parameters for all cases. The KIFMM parameters m and s represent the multipole expansion order and the maximum number of particles per leaf box, respectively. The results of our performance comparison show that the algorithm of the present paper is competitive with the KIFMM algorithm for the geometry and target accuracies considered. Fig. 8(a) shows the time taken for a matrix–vector product in each code for both accuracies. For the lower-accuracy case, our algorithm is faster for all process counts, though the KIFMM code is faster for the higher accuracy case. This is expected, as the complexity estimates for ACE and KIFMM when the number of points per box is optimized for runtime are $\mathcal{O}(NP^3)$ and $\mathcal{O}(Nm)$, respectively [14,17]. As shown in Fig. 8(b), our algorithm scales reasonably well up to 20 MPI ranks, but not quite as well as the KIFMM code. We note that the KIFMM code ran out of memory for 20 MPI ranks in high-accuracy mode.

Table 2
Parameters for ACE vs. KIFMM performance runs.

	Low-accuracy	High-accuracy
ACE P	4	13
ACE d_0	0.0625	0.08
ACE error	5.5×10^{-5}	4.5×10^{-7}
KIFMM m	4	6
KIFMM s	30	100
KIFMM error	8.7×10^{-5}	4.4×10^{-7}

5. Conclusion

In this paper, we have presented a fast, adaptive, and scalable algorithm for evaluating pair potentials involving non-oscillatory kernels of arbitrary form. We introduced controllably-accurate operators for calculating interactions arising from the non-uniform tree structure and algorithms for building, load balancing, and efficiently traversing the resulting non-uniform tree structure. We demonstrated the merits of this algorithm in terms of both error control and parallel performance through a series of numerical example involving several different kernels, including non-decaying kernels, and both uniform and extremely non-uniform distributions of up to 5 billion particles on as many as 16,384 CPU cores. The presented methods have many potential applications in large-scale computational physics, machine learning, and beyond.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has used computational resources available at the High Performance Computing Center at Michigan State University, and at the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. We also acknowledge support from the National Science Foundation, USA under grants ECCS-1408115, DMS-1619960, OAC-1807622, OAC-1822932.

Appendix A. Stokes potentials

The three Green's functions for the linearized Navier–Stokes equations in Stokes flow may also be referred to as the Stokes potentials. They are tensor-valued functions of position, given by

$$S_{ij}(\mathbf{r}) = \frac{\delta ij}{r} - \frac{r_i r_j}{r^3}, \tag{A.1a}$$

$$\Omega_{ij}(\mathbf{r}) = \epsilon_{ijk}\frac{r_k}{r^3}, \tag{A.1b}$$

$$\sigma_{ijk}(\mathbf{r}) = -6\frac{r_i r_j r_k}{r^5}, \tag{A.1c}$$

where $\{S_{ij}\}$ are the components of the Stokeslet, $\{\Omega_{ij}\}$ are the components of the rotlet, and $\{\sigma_{ijk}\}$ are the components of the stresslet. Alternatively, the above potentials can be rearranged and expressed in terms of derivatives on r [32–34]:

$$S_{ij}(\mathbf{r}) \doteq \left(\delta_{ij}\nabla^2 - \partial_i\partial_j\right)\psi(\mathbf{r}), \tag{A.2a}$$

$$\Omega_{ij}(\mathbf{r}) \doteq \left(-\epsilon_{ijk}\partial_k\nabla^2\right)\psi(\mathbf{r}), \tag{A.2b}$$

$$\sigma_{ijk}(\mathbf{r}) \doteq \left[\left(\delta_{ij}\partial_k + \delta_{jk}\partial_i + \delta_{ki}\partial_j\right)\nabla^2 - 2\partial_i\partial_j\partial_k\right]\psi(\mathbf{r}) \tag{A.2c}$$

where $\psi(\mathbf{r}) = r$, δ_{ij} is the Kronecker delta, ∇^2 denotes the Laplacian, ∂_i represents the derivative along the ith direction (from the set $\{x, y, z\}$), and ϵ_{ijk} denotes the Levi-Civita symbol. By applying the ACE algorithm to the kernel r and leveraging Eq. (4), one may obtain any of the Stokes potentials with these formulas. This is the approach taken in Section 4.

Appendix B. Accelerated cartesian expansion: Theorems

Theorem 1 (*Charge-to-Multipole (C2M)*)**.** *Assuming that k sources exist in the domain Ω_s, the potential function $\Phi(\mathbf{r})$ at any point $\mathbf{r}$ significantly away from Ω_s is given by*

$$\begin{aligned}\Phi(\mathbf{r}) &= \sum_{n=0}^{\infty}\mathbf{M}^{(n)}(\mathbf{r}_s^c)\cdot n\cdot\nabla^n\psi(\mathbf{r})\\ \mathbf{M}^{(n)}(\mathbf{r}_s^c) &= \sum_{i=1}^{k}\frac{(-1)^n}{n!}(\mathbf{r}_i-\mathbf{r}_s^c)^n u_i,\end{aligned} \tag{B.1}$$

where $\mathbf{M}^{(n)}(\mathbf{r}_s^c)$ is the rank-n multipole tensor about $\mathbf{r}_s^c$.

Next, these multipoles can be re-expressed about $\mathbf{r}_{s,p}^c$ using

Theorem 2 (*Multipole-to-Multipole (M2M)*)**.** *The multipole expansion $\mathbf{M}(\mathbf{r}_s^c)$ may be re-centered about the center $\mathbf{r}_{s,p}^c$ of a domain $\Omega_s^p \supset \Omega_s$ via*

$$\mathbf{M}_p^{(n)}(\mathbf{r}_{s,p}^c) = \sum_{m=0}^{n}\sum_{P(n,m)}\frac{m!}{n!}(\mathbf{r}_{s,p}^c-\mathbf{r}_s^c)^{n-m}\mathbf{M}^{(m)}(\mathbf{r}_s^c), \tag{B.2}$$

where $P(n, m)$ is the permutation of all partitions of n into sets of size $n-m$ and m, and $\mathbf{M}_p(\mathbf{r}_{s,p}^c)$ is the re-centered multipole expansion.

Next, we translate these multipoles about Ω_s^p to Ω_o^p.

Theorem 3 (*Multipole-to-Local (M2L)*)**.** *Assume that a domain Ω_o^p centered at $\mathbf{r}_{o,p}^c$ exists at some distance from Ω_s^p, and $\Omega_o^p \cap \Omega_s^p = \emptyset$. Then, given a multipole expansion $\mathbf{M}_p^{(n)}(\mathbf{r}_{s,p}^c)$ centered at $\mathbf{r}_{s,p}^c$, the potential $\Phi(\mathbf{r})$ may be alternatively expressed in the form*

$$\begin{aligned}\Phi(\mathbf{r}) &= \sum_{n=0}^{\infty}(\mathbf{r}-\mathbf{r}_{o,p}^c)^n\cdot n\cdot\mathbf{L}_p^{(n)}(\mathbf{r}_{o,p}^c),\\ \mathbf{L}_p^{(n)}(\mathbf{r}_{o,p}^c) &= \frac{1}{n!}\sum_{m=n}^{\infty}\mathbf{M}_p^{(m-n)}(\mathbf{r}_{s,p}^c)\cdot(m-n)\cdot\nabla^m\psi(|\mathbf{r}_{o,p}^c-\mathbf{r}_{s,p}^c|),\end{aligned} \tag{B.3}$$

where $\mathbf{L}_p(\mathbf{r}_{o,p}^c)$ is referred to as the local expansion for Ω_o^p.

Next, we map the local expansions in Ω_o^p to Ω_o.

Theorem 4 (*Local-to-Local (L2L)*)**.** *Given a local expansion $\mathbf{L}^{(n)}(\mathbf{r}_{o,p}^c)$ within the domain Ω_o^p about the center $\mathbf{r}_{o,p}^c$, the local expansion within $\Omega_o \subset \Omega_o^p$ centered at $\mathbf{r}_o^c$ is given by*

$$\mathbf{L}^{(n)}(\mathbf{r}_o^c) = \sum_{m=n}^{\infty}\binom{m}{m-n}\mathbf{L}^{(m)}(\mathbf{r}_{o,p}^c)\cdot(m-n)\cdot(\mathbf{r}_o^c-\mathbf{r}_{o,p}^c)^{m-n}. \tag{B.4}$$

Finally, the local expansions are mapped to observers.

Theorem 5 (*Local-to-Observer (L2O)*)**.** *The potential at a point $\mathbf{r}\in\Omega_o$ can be obtained from the local expansion within Ω_o centered at $\mathbf{r}_o^c$ via*

$$\Phi(\mathbf{r}) = \sum_{n=0}^{\infty}(\mathbf{r}-\mathbf{r}_o^c)^n\cdot n\cdot\mathbf{L}^{(n)}(\mathbf{r}_o^c). \tag{B.5}$$

These theorems collectively provide a framework for expressing addition theorems for any smooth and non-oscillatory kernel function.

References

[1] A. Dutt, M. Gu, V. Rokhlin, SIAM J. Numer. Anal. 33 (5) (1996) 1689–1711.
[2] T. Hofmann, B. Schölkopf, A.J. Smola, Ann. Statist. (2008) 1171–1220.
[3] R. Yokota, L.A. Barba, M.G. Knepley, Comput. Methods Appl. Mech. Engrg. 199 (25–28) (2010) 1793–1804.
[4] R. Krasny, L. Wang, SIAM J. Sci. Comput. 33 (5) (2011) 2341–2355.
[5] G.C. Linderman, M. Rachh, J.G. Hoskins, S. Steinerberger, Y. Kluger, Nat. Methods 16 (3) (2019) 243.
[6] L. Greengard, V. Rokhlin, J. Comput. Phys. 73 (2) (1987) 325–348.
[7] L. Greengard, The Rapid Evaluation of Potential Fields in Particle Systems, MIT Press, 1988.
[8] J. Barnes, P. Hut, Nature 324 (6096) (1986) 446–449.
[9] J. Song, C.-C. Lu, W.C. Chew, IEEE Trans. Antennas and Propagation 45 (10) (1997) 1488–1493.
[10] H. Cheng, W.Y. Crutchfield, Z. Gimbutas, L.F. Greengard, J.F. Ethridge, J. Huang, V. Rokhlin, N. Yarvin, J. Zhao, J. Comput. Phys. 216 (1) (2006) 300–325.
[11] M. Messner, M. Schanz, E. Darve, J. Comput. Phys. 231 (4) (2012) 1175–1196.
[12] B. Engquist, L. Ying, SIAM J. Sci. Comput. 29 (4) (2007) 1710–1737.
[13] S. Hughey, H. Aktulga, M. Vikram, M. Lu, B. Shanker, E. Michielssen, IEEE Trans. Antennas and Propagation 67 (2) (2019) 1094–1107.
[14] L. Ying, G. Biros, D. Zorin, J. Comput. Phys. 196 (2) (2004) 591–626.
[15] W. Fong, E. Darve, J. Comput. Phys. 228 (23) (2009) 8712–8725.
[16] R. Wang, C. Chen, J. Lee, E. Darve, PBBFMM3D: a parallel black-box fast multipole method for non-oscillatory kernels, 2019, arXiv preprint arXiv:1903.02153.
[17] B. Shanker, H. Huang, J. Comput. Phys. 226 (1) (2007) 732–753.
[18] M. Vikram, A. Baczewski, B. Shanker, L. Kempel, J. Comput. Phys. 229 (24) (2010) 9119–9134.
[19] A.D. Baczewski, D.L. Dault, B. Shanker, IEEE Trans. Antennas and Propagation 60 (9) (2012) 4281–4290.
[20] M. Vikram, H. Huang, B. Shanker, T. Van, IEEE Trans. Antennas and Propagation 57 (7) (2009) 2094–2104.
[21] F.A. Cruz, M.G. Knepley, L.A. Barba, Internat. J. Numer. Methods Engrg. 85 (4) (2011) 403–428.
[22] D. Potter, J. Stadel, R. Teyssier, Comput. Astrophys. Cosmol. 4 (1) (2017) 2.

[23] L. Ying, G. Biros, D. Zorin, H. Langston, Supercomputing, 2003 ACM/IEEE Conference, IEEE, 2003, p. 14.
[24] D. Malhotra, G. Biros, ACM Trans. Math. Softw. 43 (2) (2016) 17.
[25] J. Carrier, L. Greengard, V. Rokhlin, SIAM J. Sci. Stat. Comput. 9 (4) (1988) 669–686.
[26] M.S. Warren, J.K. Salmon, Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, ACM, 1993, pp. 12–21.
[27] J.P. Singh, C. Holt, J.L. Hennessy, A. Gupta, Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, ACM, 1993, pp. 54–65.
[28] J.P. Singh, C. Holt, T. Totsuka, A. Gupta, J. Hennessy, J. Parallel Distrib. Comput. 27 (2) (1995) 118–141.
[29] H. Sundar, R.S. Sampath, G. Biros, SIAM J. Sci. Comput. 30 (5) (2008) 2675–2708.
[30] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, G. Biros, Commun. ACM 55 (5) (2012) 101–109.
[31] D. Malhotra, G. Biros, Commun. Comput. Phys. 18 (3) (2015) 808–830.
[32] C. Pozrikidis, J. Eng. Math. 30 (1–2) (1996) 79–96.
[33] X.-J. Fan, N. Phan-Thien, R. Zheng, Z. Angew. Math. Phys. 49 (2) (1998) 167–193.
[34] L. af Klinteberg, D.S. Shamshirgar, A.-K. Tornberg, Res. Math. Sci. 4 (1) (2017) 1.