Computer Physics Communications 265 (2021) 108017

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

A GPU-accelerated fast multipole method based on barycentric Lagrange interpolation and dual tree traversal ☆

Leighton Wilson *,[1], Nathan Vaughn [1], Robert Krasny

Department of Mathematics, University of Michigan, Ann Arbor, MI 48109, USA

ARTICLE INFO

Article history:
Received 12 December 2020
Received in revised form 16 April 2021
Accepted 23 April 2021
Available online 4 May 2021

Keywords:
Fast summation
Barycentric Lagrange interpolation
Dual tree traversal
GPU
MPI remote memory access

ABSTRACT

We present a GPU-accelerated fast multipole method (FMM) called BLDTT, which uses barycentric Lagrange interpolation for the near-field and far-field approximations, and dual tree traversal to construct the interaction lists. The scheme replaces well-separated particle-particle interactions by adaptively chosen particle-cluster, cluster-particle, and cluster-cluster approximations given by barycentric Lagrange interpolation on a Chebyshev grid of proxy particles in each cluster. The BLDTT employs FMM-type upward and downward passes, although here they are adapted to interlevel polynomial interpolation. The BLDTT is kernel-independent, and the approximations have a direct sum form that efficiently maps onto GPUs, where targets provide an outer level of parallelism and sources provide an inner level of parallelism. The code uses OpenACC directives for GPU acceleration and MPI remote memory access for distributed memory parallelization. Computations are presented for different particle distributions, domains, and interaction kernels, and for unequal targets and sources. The BLDTT consistently outperforms our earlier particle-cluster barycentric Lagrange treecode (BLTC). On a single GPU for problem size ranging from N=1E5 to 1E8, the BLTC scales like $O(N \log N)$ and the BLDTT scales like $O(N)$. We also present MPI strong scaling results for the BLDTT and BLTC with N=64E6 particles running on 1 to 32 GPUs.

1. Introduction

Long-range particle interactions are essential in many areas of computational physics including the calculation of electrostatic and gravitational potentials as well as discrete convolution sums in boundary element methods. For a specific example consider the potential due to a set of N charged particles,

$$\phi(\mathbf{x}_i) = \sum_{j=1}^{N} G(\mathbf{x}_i, \mathbf{x}_j) q_j, \quad i = 1:N, \tag{1}$$

where $\mathbf{x}_i$ is a target particle, $\mathbf{x}_j$ is a source particle with charge q_j, and $G(\mathbf{x}, \mathbf{y})$ is the Coulomb kernel. In this work we assume the particles reside in 3D space.

The cost of evaluating the potentials $\phi(\mathbf{x}_i)$ by direct summation scales like $O(N^2)$ and several methods have been developed that reduce the cost and achieve sub-quadratic scaling. Among these, mesh-based methods such as particle-particle/particle-mesh (P3M) [1] and particle-mesh Ewald (PME) [2]) project the particles onto a regular mesh where the fast Fourier transform or multigrid can be applied. Alternatively, tree-based methods such as the treecode (TC) [3] and fast multipole method (FMM) [4] partition the particles into a hierarchical tree of clusters so that the effect of a cluster can be approximated. Other related approaches for fast summation of particle interactions include panel clustering [5], hierarchical matrices [6], and multilevel summation [7]. The present work is concerned with tree-based methods and next we outline their features.

1.1. Tree-based fast summation methods

Tree-based fast summation methods like the TC and FMM have two phases, a precompute phase in which the particles are partitioned into a hierarchical tree of clusters, and a compute phase in which the potentials are computed by approximating the effect of well-separated particles and clusters. If the particle distribution is reasonably homogeneous, the precompute phase scales like $O(N \log N)$, but with a small prefactor so that in practice it accounts for a small part of the total run time.

In the compute phase, the original TC used monopole approximations [3], while the FMM used higher order multipole approx-

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

* Corresponding author at: Department of Mathematics, University of Michigan, 530 Church St, Ann Arbor, MI 48109, USA.
E-mail addresses: lwwilson@umich.edu (L. Wilson), njvaughn@umich.edu (N. Vaughn), krasny@umich.edu (R. Krasny).

[1] Co-first authors.

https://doi.org/10.1016/j.cpc.2021.108017
0010-4655/

imations [4]. Subsequent work employed higher order Cartesian Taylor expansions in the TC [8], as well as plane wave expansions in the FMM [9]. Aside from the type of approximation being used, the scaling of the compute phase is determined by how the particles and clusters interact, and next we briefly review how this is done in each method.

The TC compute phase loops through the target particles and for each particle it traverses the tree from the root to the leaves, where well-separated clusters are identified using a multipole acceptance criterion (MAC) that depends on the cluster radius and the particle-cluster distance. When a well-separated particle-cluster pair is identified, the TC updates the target potential using a particle-cluster approximation. The TC compute phase scales like $O(N \log N)$, where the factor N represents the loop over target particles and the factor $\log N$ represents the number of levels in the tree.

The FMM compute phase has an upward pass to compute cluster moments using multipole-to-multipole translations, and a downward pass to compute potentials using multipole-to-local and local-to-local translations. The interaction list of well-separated cluster-cluster pairs is commonly defined in a uniform way at each level of the tree using a $3 \times 3 \times 3$ list of parent neighbors, although the FMM has been extended to use a $5 \times 5 \times 5$ list at the leaf level [10], as well as a more optimal multipole-to-local stencil [11]. The compute phase of the FMM scales like $O(N)$ [4,9].

The dual tree traversal (DTT) is an alternative way of building the interaction list in an FMM using a target tree and a source tree, where the two trees are traversed concurrently from the root to the leaves, and a MAC is used to identify well-separated cluster-cluster pairs [12,13]. These methods are often called DTT-FMMs and the compute phase also scales like $O(N)$ [13,14]. While the conventional FMM interaction list is favored for homogeneous particle distributions, the more flexible DTT approach is expected to be better suited for non-homogeneous distributions [13,15,16] as arise in molecular dynamics [17,18] and astrophysical simulations [19–22].

1.2. Kernel-independent methods

Many tree-based fast summation methods have employed analytic series approximations for specific kernels such as multipole expansions [4,9,23] and Cartesian Taylor expansions [24–26] for the Coulomb and Yukawa kernels. Alternative approximations for the Coulomb kernel were also investigated utilizing the Poisson integral formula [27] and multipole expansions at pseudoparticles [28]. Subsequently, kernel-independent methods were developed that require only kernel evaluations and are suitable for a large class of kernels. Among these, the kernel-independent FMM (KIFMM) uses equivalent densities defined on proxy surfaces [29], while several other methods use polynomial interpolation [30–34]. A number of related proxy point methods have recently been developed using skeletonized interpolation [35] and interpolative decomposition [36], and several kernel-independent fast summation methods have been parallelized on multicore CPU systems [37–43].

1.3. GPU implementations

A recent trend in parallel computing uses graphics processing units (GPUs) for high-throughput arithmetic. The direct sum in Eq. (1) is well suited for GPU computing because the kernel evaluations $G(\mathbf{x}_i, \mathbf{x}_j)$ are independent of each other and can be computed concurrently without thread divergence, where the targets $\mathbf{x}_i$ provide an outer level of parallelism and the sources $\mathbf{x}_j$ provide an inner level of parallelism. While GPU implementations of the direct sum achieve substantial speedup over CPU implementations [44–46], they still scale like $O(N^2)$ and there is great interest in implementing the sub-quadratic scaling tree-based methods on GPUs, although this is challenging due to their greater complexity. Previous work in this direction includes GPU implementations of the TC [47–52], FMM [11,53–57] and DTT-FMM [58,59].

1.4. Present work

Here we present a GPU-accelerated DTT-FMM called BLDTT, which uses barycentric Lagrange interpolation for the near-field and far-field approximations [60,34]. The approximations require only kernel evaluations, ensuring that the BLDTT is kernel-independent. The BLDTT employs FMM-type upward and downward passes [4], although here they are adapted to interlevel polynomial interpolation. The clusters are rectangular boxes, and a parent cluster is divided into eight, four, or two children depending the parent's aspect ratio. Each cluster has a tensor product grid of proxy particles located at Chebyshev interpolation points. The MAC employed in the DTT checks the cluster radii and their center-center distance, and it also compares the number of target and source particles to the number of proxy particles, enabling an adaptive choice between direct particle-particle interactions and three types of approximations (particle-cluster, cluster-particle, cluster-cluster). As with other DTT-FMMs, the precompute phase scales like $O(N \log N)$ with a small prefactor and the compute phase scales like $O(N)$. The BLDTT is an extension of previous related work on treecodes [34,52,61,62].

As will be seen, the approximations used in the BLDTT resemble the direct sum in Eq. (1) and can be efficiently mapped onto GPUs. Based on this observation, the BLDTT is implemented using OpenACC directives for GPU acceleration and MPI remote memory access for distributed memory parallelization. Computations are presented for different particle distributions, domains, and kernels, and for unequal targets and sources, and the BLDTT consistently outperforms our earlier barycentric Lagrange treecode (BLTC [34,52]). On a single GPU for problem size ranging from N=1E5 to 1E8, the BLTC scales like $O(N \log N)$, while the BLDTT scales like $O(N)$. We also present MPI strong scaling results for the BLDTT and BLTC using N=64E6 particles running on 1 to 32 GPUs. The BLDTT code is available on GitHub in the BaryTree library for fast summation of particle interactions [63].

The remainder of the paper is organized as follows. Section 2 describes the barycentric Lagrange dual tree traversal (BLDTT) fast multipole method. Section 3 presents our implementation of the BLDTT using OpenACC directives for GPU acceleration and MPI remote memory access for distributed memory parallelization. Section 4 describes the test cases, Section 5 presents numerical results, and Section 6 gives the conclusions. Appendices A and B derive formulas used in the upward and downward passes.

2. Description of BLDTT fast multipole method

2.1. Barycentric Lagrange interpolation

We briefly review the barycentric Lagrange form of polynomial interpolation in 1D [60]. Given a function $f(x)$ and $n+1$ distinct points $s_k, k = 0:n$, the interpolating polynomial can be written as

$$p(x) = \sum_{k=0}^{n} f(s_k) L_k(x), \tag{2}$$

where the barycentric form of the Lagrange polynomial $L_k(x)$ is

$$L_k(x) = \frac{\dfrac{w_k}{x - s_k}}{\displaystyle\sum_{k'=0}^{n} \frac{w_{k'}}{x - s_{k'}}}, \quad w_k = \frac{1}{\prod_{j=0,\, j\neq k}^{n} (s_k - s_j)}, \quad k = 0:n. \tag{3}$$

Table 1
Summary of notation, PP = particle-particle, PC = particle-cluster, CP = cluster-particle, CC = cluster-cluster.

Symbol	Description	Symbol	Description
M, N	number of target, source particles	$G(\mathbf{x}, \mathbf{y})$	interaction kernel
$\mathbf{x}_i, x_i$	target particle in 3D, 1D	$\phi(\mathbf{x}_i)$	potential at target $\mathbf{x}_i$
$\mathbf{y}_j, y_j$	source particle in 3D, 1D	$\phi_{PP}(\mathbf{x}_i, C_t, C_s)$	PP potential at target $\mathbf{x}_i$ in C_t due to sources in C_s
q_j	source charge of $\mathbf{y}_j$		
C_t, C_s	target cluster, source cluster	$\phi_{PC}(\mathbf{x}_i, C_t, \widehat{C}_s)$	PC potential at target $\mathbf{x}_i$ in C_t due to proxy sources in $\widehat{C}_s$
$\mathbf{t}_\ell, t_\ell$	target proxy particle in 3D, 1D		
$\mathbf{s}_\mathbf{k}, s_k$	source proxy particle in 3D, 1D	$\phi_{CP}(\mathbf{t}_\ell, \widehat{C}_t, C_s)$	CP proxy potential at proxy target $\mathbf{t}_\ell$ in $\widehat{C}_t$ due to sources in C_s
$\widehat{q}_k$	proxy charge of s_k		
$\widehat{C}_t, \widehat{C}_s$	proxy particles in target cluster, source cluster	$\phi_{CC}(\mathbf{t}_\ell, \widehat{C}_t, \widehat{C}_s)$	CC proxy potential at proxy target $\mathbf{t}_\ell$ in $\widehat{C}_t$ due to proxy sources in $\widehat{C}_s$

This work employs Chebyshev points of the second kind,

$$s_k = \cos\theta_k, \quad \theta_k = \pi k/n, \quad k = 0:n, \tag{4}$$

and in that case the interpolation weights are

$$w_k = (-1)^k \delta_k, \quad k = 0:n, \tag{5}$$

where $\delta_k = 1/2$ if $k = 0$ or n, and $\delta_k = 1$ otherwise [60,64]. The BLDTT uses barycentric Lagrange interpolation in 3D rectangular boxes, where the interpolation points $\mathbf{s}_\mathbf{k} = (s_{k_1}, s_{k_2}, s_{k_3})$ form a tensor product grid of Chebyshev points adapted to the box. As will be seen, the interpolation points play the role of proxy particles.

2.2. Algorithm overview

The BLDTT described below computes the potential at M target particles $\mathbf{x}_i$ due to N source particles and charges $\mathbf{y}_j, q_j$; Eq. (1) is a special case in which the targets and sources coincide. First, two trees of particle clusters are built, one for the targets and one for the sources, where each cluster is a rectangular box; clusters in the target tree are denoted C_t and clusters in the source tree are denoted C_s. The computed potential at a target particle $\phi(\mathbf{x}_i)$ has contributions from four types of interactions as determined by the dual tree traversal. The four types are direct particle-particle (PP) interactions of nearby particles, and particle-cluster (PC), cluster-particle (CP), and cluster-cluster (CC) approximations of well-separated particles and clusters.

Algorithm 1 is a high-level overview of the BLDTT. Lines 1-4 describe the input consisting of target and source particle data, interpolation degree n, MAC parameter θ, and the maximum number of particles in the leaves of each tree, M_0, N_0. Line 5 describes the output consisting of the computed potentials $\phi(\mathbf{x}_i)$. Line 6 builds the target tree and source tree containing target clusters C_t and source clusters C_s. Line 7 is the upward pass to compute proxy charges $\widehat{q}_\mathbf{k}$ at proxy particles $\mathbf{s}_\mathbf{k}$. Line 8 is the dual tree traversal to compute PP, PC, CP, and CC interactions. Line 9 is the downward pass to interpolate proxy potentials from proxy particles $\mathbf{t}_\ell$ to target particles $\mathbf{x}_i$. The steps will be described in detail below. The notation used in presenting the BLDTT is summarized in Table 1.

Algorithm 1 Barycentric Lagrange Dual Tree Traversal (BLDTT).

1: **input** target particles $\mathbf{x}_i, i = 1:M$
2: **input** source particles and charges $\mathbf{y}_j, q_j, j = 1:N$
3: **input** interpolation degree n, MAC parameter θ
4: **input** max particles per target leaf M_0, max particles per source leaf N_0
5: **output** potentials $\phi(\mathbf{x}_i), i = 1:M$
6: build target tree and source tree
7: upward pass to compute proxy charges $\widehat{q}_\mathbf{k}$ at proxy particles $\mathbf{s}_\mathbf{k}$ in source clusters
8: dual tree traversal to compute PP, PC, CP, CC interactions
9: downward pass to interpolate proxy potentials from proxy particles $\mathbf{t}_\ell$ to target particles $\mathbf{x}_i$

2.3. Tree building

The target and source trees are constructed by the same routines, described here for the target tree. The maximum number of particles per leaf is a user-specified parameter M_0. The root cluster is the minimal bounding box containing all target particles. The root is recursively divided into child clusters, terminating when a cluster contains fewer than M_0 particles. Division occurs at the midpoint of the cluster; in general the cluster is bisected in all three coordinate directions, resulting in eight child clusters, with two exceptions. First, a cluster is divided into only two or four children in order to maintain a good aspect ratio, that is, a ratio of longest to shortest side lengths no greater than $\sqrt{2}$. Second, a cluster is divided into only two or four children to avoid creating leaves with fewer than $M_0/2$ particles on average; in particular, if a cluster contains between M_0 and $2M_0$ particles, it is divided into two children, and if it contains between $2M_0$ and $4M_0$ particles, it is divided into four children. Upon creation, each cluster is shrunk to the minimal bounding box containing its particles, and a tensor product grid of Chebyshev points adapted to the box is created; these are also referred to as proxy particles. After building the trees, the BLDTT performs the upward pass, dual tree traversal, and downward pass, but before discussing these steps, the next subsection describes the four types of interactions used to compute potentials.

2.4. Four types of interactions

Fig. 1 depicts the four types of interactions between a target cluster C_t (left, blue) and a source cluster C_s (right, red), where dots are target/source particles $\mathbf{x}_i, \mathbf{y}_j$, and crosses are target/source proxy particles $\mathbf{t}_\ell, \mathbf{s}_\mathbf{k}$. Also shown are the target/source cluster radii r_t, r_s, and the target-source cluster distance R. These diagrams depict 2D versions of the interactions; in practice the particles reside in 3D and the clusters are rectangular boxes. Fig. 1 shows the four cases: (a) particles in C_t and C_s (PP), (b) particles in C_t and proxy particles in C_s (PC), (c) proxy particles in C_t and particles in C_s (CP), (d) proxy particles in C_t and C_s (CC). The interactions are described in detail below, where to simplify notation they are presented in 1D instead of 3D, for example replacing the bold 3-vector $\mathbf{x}_i$ by the non-bold scalar x_i; the extension to 3D is straightforward using tensor products.

Particle-particle interaction (Fig. 1a). The PP potential at a target particle $x_i \in C_t$ due to direct interaction with the source particles $y_j \in C_s$ is

$$\phi_{PP}(x_i, C_t, C_s) = \sum_{y_j \in C_s} G(x_i, y_j) q_j, \quad x_i \in C_t. \tag{6}$$

Particle-cluster approximation (Fig. 1b). The kernel is approximated by holding x_i fixed and interpolating with respect to y_j at the proxy particles s_k in C_s,

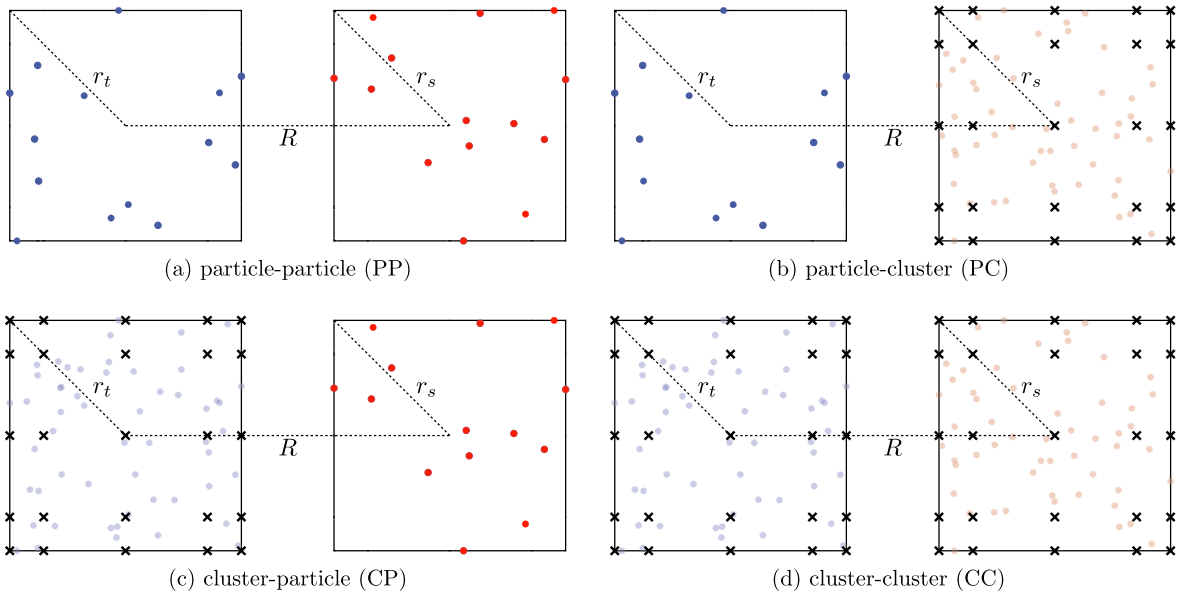


Fig. 1. Four types of interactions between a target cluster C_t (left) and a source cluster C_s (right), dots are target/source particles $\mathbf{x}_i, \mathbf{y}_j$, crosses are target/source proxy particles $\mathbf{t}_\ell, \mathbf{s}_k$, (a) direct particle-particle interaction (PP), (b) particle-cluster approximation (PC), (c) cluster-particle approximation (CP), (d) cluster-cluster approximation (CC), target/source cluster radii r_t, r_s, target-source cluster distance R. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$$G(x_i, y_j) \approx \sum_{k=0}^{n} G(x_i, s_k) L_k(y_j), \quad x_i \in C_t, \quad y_j \in C_s. \tag{7}$$

Substituting into the PP interaction Eq. (6) and rearranging terms yields the PC potential,

$$\phi_{PC}(x_i, C_t, \widehat{C}_s) = \sum_{k=0}^{n} G(x_i, s_k) \widehat{q}_k, \quad x_i \in C_t, \tag{8}$$

where the proxy charges $\widehat{q}_k$ of the proxy particles s_k are

$$\widehat{q}_k = \sum_{y_j \in C_s} L_k(y_j) q_j, \quad k = 0 : n. \tag{9}$$

Equation (8) uses $C_t, \widehat{C}_s$ to indicate that the target particles x_i interact with the proxy source particles s_k.

Cluster-particle approximation (Fig. 1c). The kernel is approximated by interpolating with respect to x_i at the proxy particles t_ℓ in C_t and holding y_j fixed,

$$G(x_i, y_j) \approx \sum_{\ell=0}^{n} G(t_\ell, y_j) L_\ell(x_i), \quad x_i \in C_t, \quad y_j \in C_s. \tag{10}$$

Substituting into the PP interaction Eq. (6) and rearranging terms yields the CP potential,

$$\phi_{CP}(x_i, \widehat{C}_t, C_s) = \sum_{\ell=0}^{n} \phi(t_\ell, \widehat{C}_t, C_s) L_\ell(x_i), \quad x_i \in C_t, \tag{11}$$

where the CP proxy potential $\phi(t_\ell, \widehat{C}_t, C_s)$ is

$$\phi(t_\ell, \widehat{C}_t, C_s) = \sum_{y_j \in C_s} G(t_\ell, y_j) q_j, \quad t_\ell \in \widehat{C}_t. \tag{12}$$

Equation (11) and Eq. (12) use $\widehat{C}_t, C_s$ to indicate that the proxy target particles t_ℓ interact with the source particles y_j. Note that Eq. (11) interpolates from the proxy target particles t_ℓ to the target particles x_i.

Cluster-cluster approximation (Fig. 1d). The kernel is interpolated with respect to x_i at the proxy particles t_ℓ in C_t and with respect to y_j at the proxy particles s_k in C_s,

$$G(x_i, y_j) \approx \sum_{k=0}^{n} \sum_{\ell=0}^{n} G(t_\ell, s_k) L_\ell(x_i) L_k(y_j), \quad x_i \in C_t, \quad y_j \in C_s. \tag{13}$$

Substituting into the PP interaction Eq. (6) and rearranging terms yields the CC potential,

$$\phi_{CC}(x_i, \widehat{C}_t, \widehat{C}_s) = \sum_{\ell=0}^{n} \phi(t_\ell, \widehat{C}_t, \widehat{C}_s) L_\ell(x_i), \quad x_i \in C_t, \tag{14}$$

where the CC proxy potential $\phi(t_\ell, \widehat{C}_t, \widehat{C}_s)$ is

$$\phi(t_\ell, \widehat{C}_t, \widehat{C}_s) = \sum_{k=0}^{n} G(t_\ell, s_k) \widehat{q}_k, \quad t_\ell \in \widehat{C}_t, \tag{15}$$

and the proxy charges $\widehat{q}_k$ were defined in Eq. (9). Equation (14) and Eq. (15) use $\widehat{C}_t, \widehat{C}_s$ to indicate that the proxy target particles t_ℓ interact with the proxy source particles s_k. Note that Eq. (14) interpolates from the proxy target particles t_ℓ to the target particles x_i.

While the PP interaction in Eq. (6) is a direct sum involving kernel evaluations of target and source particles, it should be noted that the PC, CP, CC interactions in Eqs. (8), (12), (15) also have a direct sum form involving kernel evaluations of target and source particles or proxy particles. This enables the efficient GPU implementation discussed below.

The following three subsections describe the rest of Algorithm 1 comprising the upward pass, dual tree traversal, and downward pass. Before proceeding, note that in computing the potentials $\phi(x_i)$, contributions from the $\phi_{PP}(x_i, C_t, C_s)$ and $\phi_{PC}(x_i, C_t, \widehat{C}_s)$ potentials in Eqs. (6) and (8) are included as soon as they are computed in the DTT, while contributions from the $\phi_{CP}(x_i, \widehat{C}_t, C_s)$ and $\phi_{CC}(x_i, \widehat{C}_t, \widehat{C}_s)$ potentials in Eqs. (12) and Eq. (15) are computed in the DTT and included in the downward pass.

2.5. Upward pass

The upward pass computes the proxy charges $\widehat{q}_k$ in Eq. (9) for the proxy particles s_k in each source cluster $\widehat{C}_s$ in the source tree; these are required for the PC and CC approximations in Eqs. (8)

and (15). Each source particle y_j contributes to the proxy charges $\widehat{q}_k$ of exactly one cluster at each level of the tree. Hence with N source particles and tree depth $O(\log N)$, computing the proxy charges directly by Eq. (9) requires $O(N \log N)$ operations.

Following the FMM [4], the BLDTT uses an alternative approach. Let C_s denote a parent cluster and let $C_s^i, i = 1:2$ denote the two child clusters; recall that here we consider a 1D system for clarity of presentation and the same considerations hold in 3D where the clusters have eight, four, or two children. Appendix A derives the relation

$$\widehat{q}_k = \sum_{i=1}^{2} \sum_{k_i=0}^{n} L_k(s_{k_i}) \widehat{q}_{k_i}, \quad k = 0:n, \tag{16}$$

where $L_k(x)$ are the Lagrange polynomials of the parent C_s, and $s_{k_i}, \widehat{q}_{k_i}$ are the interpolation points and proxy charges of the children C_s^i. Equation (16) shows that the proxy charges of the parent $\widehat{q}_k$ can be computed from the proxy charges of the children $\widehat{q}_{k_i}$. The upward pass starts by computing the proxy charges of the leaves of the source tree using Eq. (9) and it ascends to the root by Eq. (16). This is analogous to the upward pass in the FMM [4] where the multipole moments of a parent cluster are obtained from the moments of its children.

Computing the proxy charges this way requires $O(N)$ operations, which can be seen as follows. We briefly revert to 3D. First, computing the proxy charges for the leaves by Eq. (9) requires $O(n^3 N)$ operations because each of the N source particles contributes to one leaf and each leaf contains $O(n^3)$ proxy particles. Then each application of the child-to-parent relation Eq. (16) requires $O(n^6)$ operations, and since there are $O(N)$ parent clusters in the tree, ascending the tree requires an additional $O(n^6 N)$ operations. Combining these it follows that the operation count for the BLDTT upward pass is $O(N)$.

2.6. Dual tree traversal

The DTT determines which pairs of clusters in the target and source trees interact by one of the four options described above (PP, PC, CP, CC). Before the traversal starts, two sets of potentials are initialized to zero, potentials $\phi(x_i)$ at the target particles and proxy potentials $\phi(t_\ell)$ at the proxy target particles in each target cluster. During the traversal, the potentials $\phi(x_i)$ are incremented due to PP and PC interactions, and the proxy potentials $\phi(t_\ell)$ are incremented due to CP and CC interactions. Following the DTT, the proxy potentials $\phi(t_\ell)$ are interpolated to the target particles x_i and combined with the potentials $\phi(x_i)$ in the downward pass.

The dual tree traversal uses the recursive procedure DTT(C_t, C_s) in Algorithm 2, which takes a target cluster C_t and a source cluster C_s as input. Initially the procedure is called for the root clusters of the target and source trees. In what follows, the clusters are defined to be well-separated if $(r_t + r_s)/R < \theta$, where r_t, r_s are the target and source cluster radii and R is the center-center distance between the clusters.

If C_t and C_s are well-separated (line 2), then they interact in one of four ways depending on the number of particles they contain relative to the number of proxy particles in a cluster, which is denoted by $n_p = (n+1)^3$ in 3D. If C_t and C_s are both large (lines 3-4), then the proxy potentials are incremented by CC, Eq. (15); else if C_t is large and C_s is small (lines 5-6), then the proxy potentials are incremented by CP, Eq. (12); else if C_t is small and C_s is large (lines 7-8), then the potentials are incremented PC, Eq. (8); else C_t and C_s are both small (line 9) and the potentials are incremented PP, Eq. (6).

If C_t and C_s are not well-separated, then the traversal continues as follows. If C_t and C_s are leaves (lines 11-12), then the potentials are incremented by PP, Eq. (6). Otherwise if C_s is a leaf, then it interacts recursively with the children of C_t (line 13), while if C_t is a leaf, then it interacts recursively with the children of C_s (line 14). Finally if C_t and C_s are both not leaves, then the smaller cluster interacts recursively with the children of the larger cluster (lines 15-17).

Algorithm 2 Dual Tree Traversal.

```
1: procedure DTT(target cluster C_t, source cluster C_s)
2: if (r_t + r_s)/R < θ then
3:    if |C_t| > n_p and |C_s| > n_p then
4:       increment proxy potentials φ(t_ℓ) += φ(t_ℓ, Ĉ_t, Ĉ_s) by CC, Eq. (15)
5:    else if |C_t| > n_p and |C_s| ≤ n_p then
6:       increment proxy potentials φ(t_ℓ) += φ(t_ℓ, Ĉ_t, C_s) by CP, Eq. (12)
7:    else if |C_t| ≤ n_p and |C_s| > n_p then
8:       increment potentials φ(x_i) += φ(x_i, C_t, Ĉ_s) by PC, Eq. (8)
9:    else increment potentials φ(x_i) += φ(x_i, C_t, C_s) by PP, Eq. (6)
10: else
11:   if C_t and C_s are leaves then
12:      increment potentials φ(x_i) += φ(x_i, C_t, C_s) by PP, Eq. (6)
13:   else if C_s is a leaf then for each child C'_t of C_t do DTT(C'_t, C_s)
14:   else if C_t is a leaf then for each child C'_s of C_s do DTT(C_t, C'_s)
15:   else
16:      if |C_t| > |C_s| then for each child C'_t of C_t do DTT(C'_t, C_s)
17:      else for each child C'_s of C_s do DTT(C_t, C'_s)
```

The DTT yields potentials $\phi(x_i)$ due to PP and PC interactions and proxy potentials $\phi(t_\ell)$ due to CP and CC interactions. In the case of N homogeneously distributed source and target particles, the operation count of the dual tree traversal is $O(N)$ [13,14].

2.7. Downward pass

At this point each target cluster has proxy potentials $\phi(t_\ell)$ that were computed in the DTT by CP and CC interactions in Eqs. (12) and (15). The downward pass interpolates these proxy potentials to the target particles x_i and increments the potentials $\phi(x_i)$ that were computed in the DTT by PP and PC interactions in Eqs. (6) and (8). This can be done in two ways as described below.

First note that each target particle x_i is contained in a chain of target clusters,

$$x_i \in C_t^1 \subset C_t^2 \subset \cdots \subset C_t^L, \tag{17}$$

where the superscript denotes the level in the target tree; level 1 contains the leaves and level L is the root. In the downward pass, each target cluster C_t^m in the chain contributes its proxy potentials $\phi(t_{k_m}^m)$ to $\phi(x_i)$,

$$\phi(x_i) \mathrel{+}= \sum_{m=1}^{L} \sum_{k_m=0}^{n} \phi(t_{k_m}^m) L_{k_m}^m(x_i), \tag{18}$$

where $t_{k_m}^m$ are the proxy particles and $L_{k_m}^m(x)$ are the Lagrange polynomials of C_t^m, and $+=$ indicates that the results are combined with the potentials $\phi(x_i)$. In Eq. (18) the inner sum interpolates proxy potentials from the proxy particles $t_{k_m}^m$ to the target particle x_i, and the outer sum accumulates the results from each level in the chain. Computing $\phi(x_i)$ directly by Eq. (18) requires $O(M \log M)$ operations; the factor M is the number of target particles x_i, the factor $\log M$ is the number of levels in the target tree, and the operation count for the inner sum at each level is independent of M (it is $O(n)$ in 1D and $O(n^3)$ in 3D).

The procedure just described interpolates from the proxy particles t_m^k at each level in the target tree directly to the target particle x_i. Again following the FMM [4], the BLDTT utilizes a recursive alternative. In what follows, C_t^m is a parent cluster at level m and C_t^{m-1} is a child cluster at level $m-1$ in the chain given by Eq. (17).

The recursive procedure interpolates the parent proxy potentials $\phi(t^m_{k_m})$ to the child proxy particles $t^{m-1}_{k_{m-1}}$,

$$\phi(t^{m-1}_{k_{m-1}}) \mathrel{+}= \sum_{k_m=0}^{n} \phi(t^m_{k_m}) L^m_{k_m}(t^{m-1}_{k_{m-1}}), \tag{19}$$

where $+=$ indicates that the results are combined with the child proxy potentials $\phi(t^{m-1}_{k_{m-1}})$ due to CP and CC interactions previously computed in the DTT. The procedure starts at the root of the target tree and descends to the leaves, where the proxy potentials $\phi(t^1_{k_1})$ are interpolated to the target particles x_i and combined with the PP and PC potentials previously computed in the DTT,

$$\phi(x_i) \mathrel{+}= \sum_{k_1=0}^{n} \phi(t^1_{k_1}) L^1_{k_1}(x_i). \tag{20}$$

It is shown in Appendix B that Eq. (18) for $\phi(x_i)$ is equivalent to the combination of Eqs. (19) and (20).

Note that the parent-to-child interpolation in Eq. (19) requires $O(n^6)$ operations in 3D for each parent cluster, and the tree contains $O(M)$ parent clusters, so interpolating from the root to the leaves requires $O(n^6 M)$ operations. Note also that the proxy-to-particle interpolation in Eq. (20) requires $O(n^3)$ operations for each target particle, so interpolating from leaf proxy particles to target particles requires $O(n^3 M)$ operations. Combining these it follows that the operation count for the BLDTT downward pass is $O(M)$.

2.8. Description of BLTC

We briefly describe our previous particle-cluster barycentric Lagrange treecode (BLTC) [34,52] which has an algorithmic structure resembling the Barnes-Hut treecode [3]. Unlike the BLDTT which builds a tree on both the source and target particles, the BLTC builds a tree of clusters on the source particles and a set of batches on the target particles, where the batches correspond to the leaves of a target tree. Once the source tree is built, the BLTC computes the proxy charges for each source cluster directly from the source particles by Eq. (9). For each target batch the source tree is traversed, starting at the root and checking whether the target batch and a given source cluster are well-separated. If they are well-separated and the cluster contains more particles than interpolation points, then the batch and cluster interact by the PC approximation in Eq. (8). If they are not well-separated, then the batch interacts with the children of the cluster. The PP interaction in Eq. (6) is carried out in the remaining cases (leaves in the source tree that are not well-separated from the target batch, and source clusters that are well-separated from the target batch but have fewer particles than interpolation points). For M target particles and N source particles, the BLTC operation count is $O(N \log N)$ + $O(M \log N)$, where the first term is due to building the source tree and computing the proxy charges, and the second term is due to traversing the source tree for each target batch.

3. GPU implementation of BLDTT

The BLDTT implementation for multiple GPUs is similar to our previous BLTC implementation [52] and is available in the BaryTree library [63]. The code uses OpenACC directives for GPU acceleration and MPI remote memory access for distributed memory parallelization, where each GPU corresponds to one MPI rank. The DTT creates four interaction lists for each target cluster, one list for each type of interaction (PP, PC, CP, CC); this is done on the CPU, and once the lists are available, the interactions are computed on the GPU. This delegation of tasks enhances efficiency because the GPU can cycle through the lists rapidly and the GPU compute kernels can be queued asynchronously as described below. Hence the CPU is responsible for tree building, computing interaction lists by DTT, and MPI communication of particles and clusters, while the GPU is responsible for the upward pass, PP/PC/CP/CC computations, and the downward pass. Next we describe further details of the implementation.

3.1. MPI distributed memory parallelization

We use locally essential trees (LET) to implement distributed memory parallelization [65]. The particles are partitioned by recursive coordinate bisection into compact domains on each MPI rank using Trilinos Zoltan [66,67]. Each MPI rank builds the local source tree and local target tree for its particles. The LET of a rank is the union of the rank's local source tree and all source clusters from remote ranks that interact with its local target tree. Although building the LETs requires an all-to-all communication, the amount of data acquired by each rank grows only logarithmically with the problem size [65]. The computing and communication required to build the LETs is done using MPI passive target synchronization remote memory access (RMA). RMA is a one-sided communication model within MPI in which an origin process can *put* data onto a target process or *get* data from a target process through specially declared memory *windows*, with no active involvement from the target process. This enables each rank to construct its LET asynchronously from other ranks. We note that MPI remote memory access has previously been used in a multicore CPU implementation of the FMM [68].

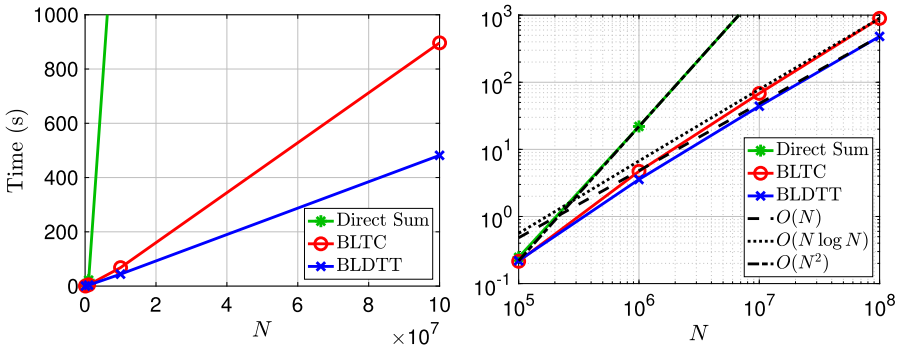
3.2. GPU compute kernels

The GPU implementation of the BLDTT employs eight compute kernels, two for the upward pass, four for computing interactions determined by the DTT, and two for the downward pass. The compute kernels are generated by OpenACC directives and are compiled with the PGI C compiler. The kernels are launched asynchronously in multiple GPU streams to hide latency as much as possible. The approach generalizes to multiple GPUs in a straightforward manner, in which each GPU corresponds to one MPI rank.

The first upward pass kernel computes the proxy charges for a given leaf in the source tree. For each leaf, the kernel is launched asynchronously and further computation is blocked until all of the leaf's proxy charges are computed. The second upward pass kernel computes the proxy charges of parent clusters using the proxy charges of the children. For a given level of the source tree above the leaves, this kernel is launched asynchronously for each cluster at that level, and further computation is blocked until all proxy charges at that level are computed.

The four DTT kernels compute the interaction of a target cluster with a source cluster. Each PP, PC, CP, and CC interaction launches one compute kernel. All such kernels are launched asynchronously and further computation is blocked until they complete. The four interaction kernels have a similar structure, with an outer loop over the particles or proxy particles in the target cluster, and an inner loop over the particles or proxy particles in the source cluster. Due to the form of barycentric Lagrange interpolation, the inner loop iterations are independent of each other and can be computed concurrently, unlike alternative approximation methods that are sequential. The outer loop is mapped to the `gang` construct in OpenACC and the inner loop is mapped to the `vector` construct. Conceptually, a member of a `gang` corresponds to a thread block and a member of a `vector` corresponds to an individual thread.

The two downward pass compute kernels are similar in structure to the upward pass kernels. At each level of the target tree above the leaves, beginning with the root, the first downward pass compute kernel is launched asynchronously for each target cluster



(a) linear scale (b) logarithmic scale

Fig. 2. Comparison of BLDTT and BLTC, run time (s) versus number of particles N=1E5, 1E6, 1E7, 1E8, random uniformly distributed particles in $[-1,1]^3$ interacting by the Coulomb kernel, MAC parameter $\theta = 0.7$, degree $n = 8$ yielding 7-8 digit accuracy, direct sum (green), BLTC (red), BLDTT (blue), (a) linear scale, (b) logarithmic scale, scaling lines $O(N^2)$ (dash-dotted), $O(N \log N)$ (dotted), $O(N)$ (dashed), computed on one NVIDIA P100 GPU.

at that level to interpolate the proxy potentials of the cluster to its children. Further computation is blocked until all compute kernels at a given level complete. Finally, the second downward pass compute kernel is launched asynchronously for each target leaf to interpolate the proxy potentials to the target particles.

4. Description of test cases

First we examine the scaling of the BLDTT and BLTC for problem size ranging from N=1E5 to 1E8, and then we demonstrate the speedup of the GPU implementation of the BLDTT over a CPU implementation. Next the BLDTT is applied to several test cases comprising different random particle distributions (uniform, Gaussian, Plummer [69,70]), domains (thin slab, square rod, spherical surface), kernels (regularized Coulomb, Yukawa, oscillatory), and for unequal targets and sources. The benefit of including CP and PC interactions in the BLDTT is verified, and the peak memory usage of the BLDTT is reported in comparison with the BLTC and direct summation. The results mentioned so far were obtained on one GPU, and finally we demonstrate the MPI strong scaling of the BLDTT on 1 to 32 GPUs.

All runs use maximum leaf and batch size $M_0 = N_0 = 2000$. The target and source particles are identical except as indicated in Section 5.5, and the source charges q_j are random and uniformly distributed on $[-1,1]$ except for the Plummer distribution where the charges are set to $1/N$. The Coulomb kernel is used in all runs except as indicated in Section 5.6. The calculations are done in double precision arithmetic and we report the relative ℓ_2 error,

$$E = \left(\sum_{i=1}^{M} (\phi_i^{ds} - \phi_i^{fs})^2 \Big/ \sum_{i=1}^{M} (\phi_i^{ds})^2 \right)^{1/2}, \tag{21}$$

where ϕ_i^{ds} are the potentials computed by direct summation and ϕ_i^{fs} are computed by fast summation (BLDTT, BLTC). The error was sampled at a random subset of 0.1% of the target particles. To facilitate comparison of the BLDTT and BLTC across the test cases presented, Figs. 4, 5, 8, 10 use the same axes to display the run time versus error.

The computations were performed on the NVIDIA P100 GPU nodes at the San Diego Supercomputer Center Comet machine, where each node has four GPUs and each GPU has 16GB of memory. These resources were provided through the Extreme Science and Engineering Discovery Environment (XSEDE) [71]. The examples directory of the BaryTree library contains the executable (`random_cube_reproducible`) used to run the BLDTT and BLTC [63]. The code was written in C and compiled with the PGI C compiler using the `-O3` optimization flag.

Table 2
Comparison of BLDTT and BLTC, number of particles $N =$ 1E5, 1E6, 1E7, 1E8, random uniformly distributed particles in $[-1,1]^3$ interacting by the Coulomb kernel, MAC parameter $\theta = 0.7$, degree $n = 8$, run time (s) from Fig. 2, relative ℓ_2 error, computed on one NVIDIA P100 GPU.

N	BLTC		BLDTT	
	Time (s)	Error	Time (s)	Error
1E5	2.15E−1	1.75E−8	2.19E−1	1.58E−8
1E6	4.71E+0	1.42E−7	3.56E+0	3.67E−8
1E7	6.81E+1	4.68E−7	4.40E+1	4.12E−8
1E8	8.96E+2	9.23E−7	4.82E+2	4.17E−8

5. Results

5.1. Scaling with problem size

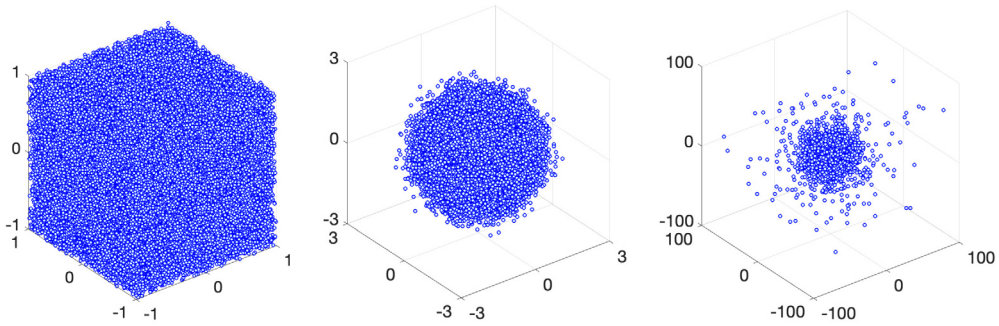
Fig. 2 shows the run time (s) for direct summation (green), BLTC (red), and BLDTT (blue) with N=1E5, 1E6, 1E7, 1E8 random uniformly distributed particles in the $[-1,1]^3$ cube interacting by the Coulomb kernel. The BLDTT and BLTC use MAC parameter $\theta = 0.7$ and interpolation degree $n = 8$, yielding 7-8 digit accuracy. Fig. 2(a) is a linear plot, showing that the BLDTT is about twice as fast as the BLTC, and both are much faster than direct summation. Fig. 2(b) is a logarithmic plot with reference lines scaling as $O(N)$ (dashed), $O(N \log N)$ (dotted), and $O(N^2)$ (dash-dotted), showing that as the problem size increases, the BLTC has asymptotic $O(N \log N)$ scaling, while the BLDTT has asymptotic $O(N)$ scaling as expected. Table 2 records the run time and error; the asymptotic scaling of the run time can be quantitatively verified, and although the error increases slightly with problem size, the BLDTT error is consistently smaller than the BLTC error.

5.2. GPU acceleration of BLDTT

We compare the BLDTT running on one NVIDIA P100 GPU and on 8 CPU cores (Intel Xeon E5-2680v3 processor, 2.50 GHz, MPI parallelization). Table 3 gives the run times showing that the BLDTT achieves 30–40× speedup on the GPU compared to 8 CPU cores. The BLDTT errors are the same as in Table 2.

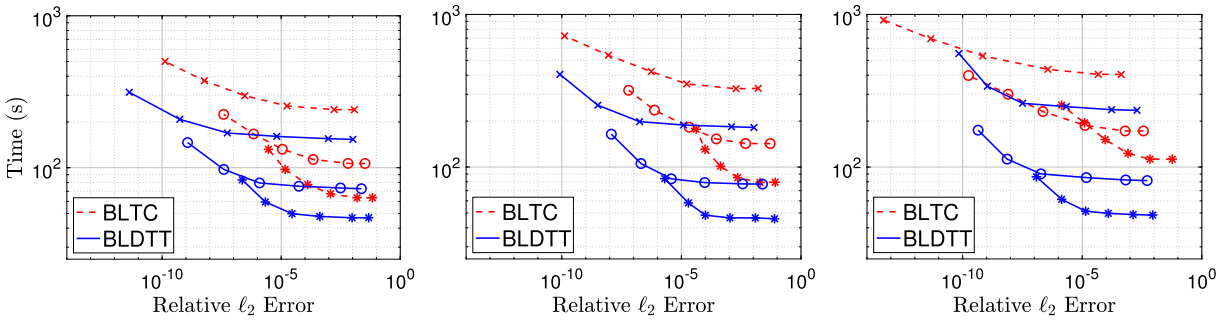
5.3. Different random particle distributions

We examine the BLDTT performance for three random particle distributions: (a) uniform in $[-1,1]^3$, (b) Gaussian with radial pdf $\frac{1}{\sqrt{6\pi}} \exp\left(-r^2/6\right)$, (c) Plummer [69,70] with radial pdf $\frac{3}{4\pi}\left(1+r^2\right)^{-5/2}$ and cutoff at ± 100 in all three Cartesian coordinates. The charges of the uniform and Gaussian particles are uniformly distributed in $[-1,1]$, and the Plummer particle charges



(a) uniform (b) Gaussian (c) Plummer

Fig. 3. Different random particle distributions, N=4E5, (a) uniform, (b) Gaussian, (c) Plummer.



(a) uniform (b) Gaussian (c) Plummer

Fig. 4. Different random particle distributions, run time (s) versus relative ℓ_2 error, N=2E7, (a) uniform, (b) Gaussian, (c) Plummer, BLTC (red, dashed), BLDTT (blue, solid), connected curves represent constant MAC θ (0.5 $\times$; 0.7 $\circ$; 0.9 $*$), interpolation degree $n = 1, 2, 4, 6, 8, 10$ increases from right to left on each curve, computed on one NVIDIA P100 GPU.

Table 3
Comparison of BLDTT running on one NVIDIA P100 GPU and on 8 CPU cores, number of particles $N =$ 1E5, 1E6, 1E7, 1E8, random uniformly distributed particles in $[-1, 1]^3$ interacting by the Coulomb kernel, MAC parameter $\theta = 0.7$, degree $n = 8$, run time (s), speedup, same errors as in Table 2.

N	CPU Time (s)	GPU Time (s)	Speedup
1E5	7.84E+0	2.19E−1	35.8
1E6	1.45E+2	3.56E+0	40.7
1E7	1.40E+3	4.40E+1	31.8
1E8	1.70E+4	4.82E+2	35.3

are set to $1/N$, where N is the number of particles. Fig. 3 depicts the three distributions with N=4E5. Compared to the uniform case (a), the Gaussian and Plummer distributions (b,c) are concentrated near the origin, with the Gaussian decaying more rapidly away from the origin and the Plummer decaying more slowly.

Fig. 4 shows the run time (s) versus relative ℓ_2 error for the BLDTT (blue, solid) and BLTC (red, dashed) on these three distributions with N=2E7. Each connected curve represents constant MAC with $\theta = 0.5$ ($\times$), $\theta = 0.7$ ($\circ$), $\theta = 0.9$ ($*$), and the interpolation degree $n = 1, 2, 4, 6, 8, 10$ increases from right to left on each curve. For these parameter choices the errors span the range from 1 digit to 10 digit accuracy. Large θ is more efficient for low accuracy and small θ is more efficient for high accuracy. The BLDTT has consistently better performance than the BLTC and is less sensitive to non-uniformity in the distribution.

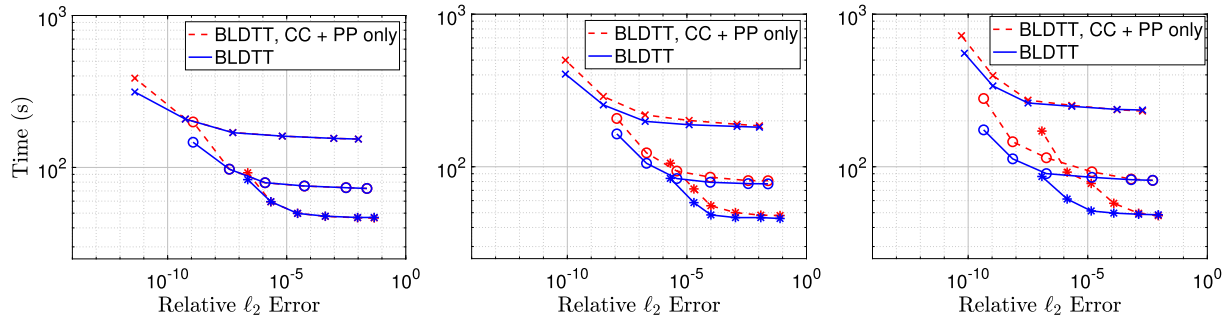
To demonstrate the benefit of including PC and CP interactions, Fig. 5 compares two versions of the BLDTT, the one presented in this work using PP, PC, CP and CC interactions (blue, solid), and one using only CC and PP interactions (red, dashed). The results show that the first version has consistently better performance, especially for the non-uniform Gaussian and Plummer distributions. When only CC and PP interactions are used, the interaction between a target cluster and a source cluster is handled by PP interaction if either cluster has fewer particles than interpolation points, whereas the flexibility to use PC or CP interactions in those cases yields better performance.

To further examine the effect of including PC and CP interactions, next we compare the number of pointwise interactions used by the two versions of the BLDTT, where a pointwise interaction refers to one kernel evaluation $G(\mathbf{x}, \mathbf{y})$. Results are shown for MAC $\theta = 0.9$ and interpolation degree $n = 1, 2, 4, 6, 8, 10$, for the same three random distributions with N=2E7 as above. Fig. 6 displays results for the four types of interactions in stacked bars, CC (blue), PP (orange), PC (yellow), CP (purple), from bottom to top, where the left bar in each pair is the BLDTT with CC and PP interactions only, and the right bar is the BLDTT with PP, PC, CP, and CC interactions. In this case a direct sum calculation would use 4E14 PP interactions, whereas the BLDTT calculations use less than 6E12 interactions. The results show that for high degree, including PC and CP interactions in the BLDTT reduces the number of PP interactions, replacing them with a smaller number of PC and CP interactions, and this effect is stronger for the non-uniform distributions.

5.4. Non-cubic domains

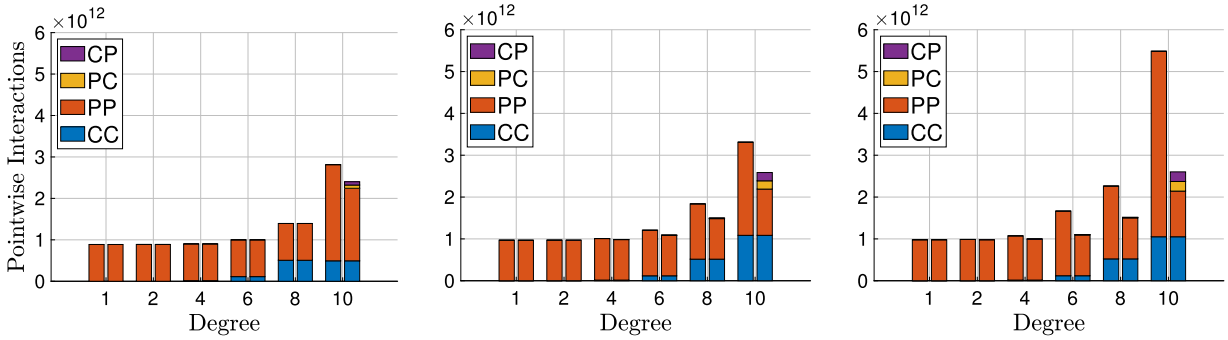
We demonstrate the BLDTT performance on three non-cubic domains depicted in Fig. 7: (a) thin slab of dimensions $1 \times 10 \times 10$, (b) square rod of dimensions $1 \times 1 \times 10$, and (c) spherical surface of radius 1. In all cases the particles are random uniformly distributed.

Fig. 8 shows the run time (s) versus relative ℓ_2 error for the BLDTT (blue, solid) and BLTC (red, dashed) with N=2E7, using MAC $\theta = 0.5, 0.7, 0.9$ and interpolation degree $n = 1, 2, 4, 6, 8, 10$. The BLDTT has consistently better performance than the BLTC. Com-



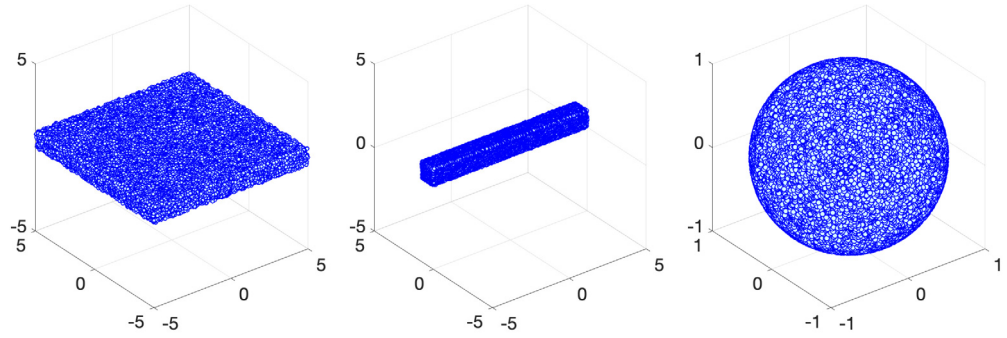
(a) uniform (b) Gaussian (c) Plummer

Fig. 5. Different random particle distributions, run time (s) versus relative ℓ_2 error, N=2E7 particles, (a) uniform, (b) Gaussian, (c) Plummer, BLDTT with only CC and PP interactions (red, dashed), BLDTT with PP, PC, CP, and CC interactions (blue, solid), connected curves represent constant MAC θ (0.5 $\times$; 0.7 $\circ$; 0.9 $*$), interpolation degree $n = 1, 2, 4, 6, 8, 10$ increases from right to left on each curve, computed on one NVIDIA P100 GPU.



(a) uniform, $\theta = 0.9$ (b) Gaussian, $\theta = 0.9$ (c) Plummer, $\theta = 0.9$

Fig. 6. Different random particle distributions, number of pointwise interactions (kernel evaluations $G(\mathbf{x}, \mathbf{y})$), N=2E7, (a) uniform, (b) Gaussian, (c) Plummer, MAC $\theta = 0.9$, interpolation degree $n = 1, 2, 4, 6, 8, 10$, each pair of bars shows BLDTT with CC and PP only (left) and BLDTT with PP, PC, CP, CC (right), direct sum calculation would use 4E14 PP interactions.



(a) thin slab (b) square rod (c) spherical surface

Fig. 7. Non-cubic domains, N=4E5 random uniformly distributed particles, (a) thin slab of dimensions $1 \times 10 \times 10$, (b) square rod of dimensions $1 \times 1 \times 10$, (c) spherical surface of radius 1.

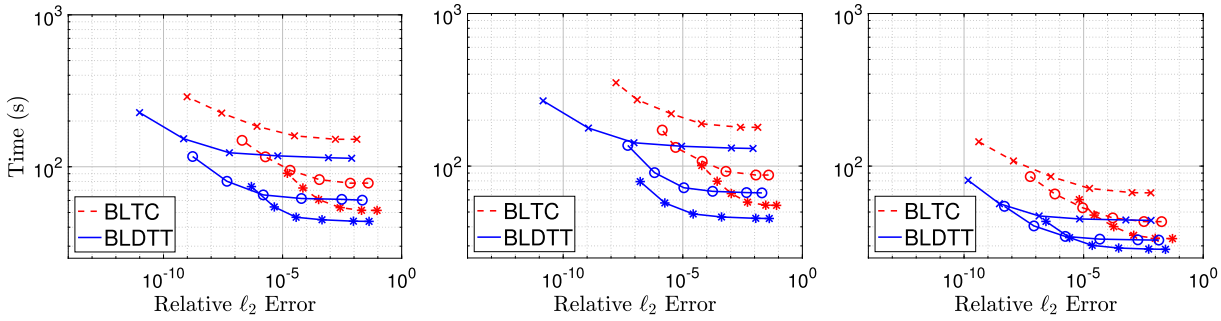


Fig. 8. Non-cubic domains, N=2E7 random uniformly distributed particles, (a) thin slab, (b) square rod, (c) spherical surface, run time (s) versus relative ℓ_2 error, BLTC (red, dashed), BLDTT (blue, solid), connected curves represent constant MAC θ (0.5 $\times$; 0.7 $\circ$; 0.9 $*$), interpolation degree $n = 1, 2, 4, 6, 8, 10$ increases from right to left on each curve, computed on one NVIDIA P100 GPU.

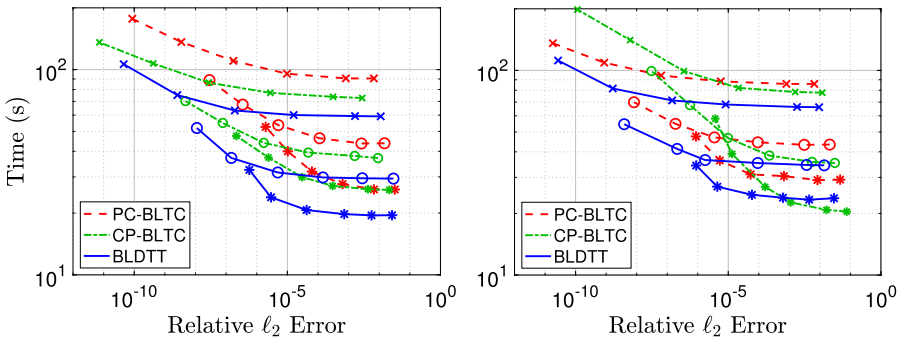


Fig. 9. Unequal targets and sources, (a) M=2E7 targets, N=2E6 sources, (b) M=2E6 targets, N=2E7 sources, random uniformly distributed particles, run time (s) versus relative ℓ_2 error, PC-BLTC (red, dashed), CP-BLTC (green, dash-dotted), BLDTT (blue, solid), connected curves represent constant MAC θ (0.5 $\times$; 0.7 $\circ$; 0.9 $*$), interpolation degree $n = 1, 2, 4, 6, 8, 10$ increases from right to left on each curve, computed on one NVIDIA P100 GPU.

pared to the cubic domain results in Fig. 4(a), the BLDTT achieves similar error and runs somewhat faster for the non-cubic domains. Recall that the tree building procedure checks the aspect ratio of the particle clusters and divides them into eight, four or two children. This yields an oct-tree for the cubic domain, and essentially a quad-tree for the thin slab and spherical surface, and a binary tree for the square rod. With fewer branches the tree is traversed faster. Recall also that each cluster is shrunk to the minimal bounding box containing its particles. These features enable the BLDTT to adapt to non-cubic domains without explicit reprogramming.

5.5. Unequal targets and sources

Next the BLDTT is applied to systems with M targets and N sources, where $M \neq N$. In addition to the particle-cluster BLTC described in Section 2.8, referred to in this section as PC-BLTC, we also consider a cluster-particle version (CP-BLTC) which builds a tree of clusters on the M targets and a set of batches on the N sources, and uses PP and CP interactions [62]. Instead of an $O(N \log N)$ upward pass to compute proxy charges, the CP-BLTC has an $O(M \log M)$ downward pass to interpolate proxy potentials to targets. While the compute phase of the PC-BLTC is $O(M \log N)$, the compute phase of the CP-BLTC is $O(N \log M)$, hence the PC-BLTC is expected to perform better when $N > M$, and the CP-BLTC is expected to perform better when $M > N$.

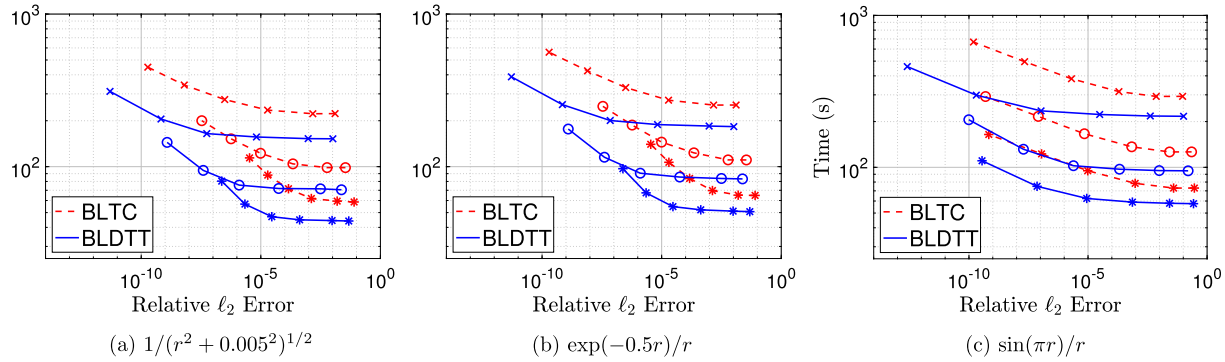
Fig. 9 shows the run time (s) versus relative ℓ_2 error for the PC-BLTC (red, dashed), CP-BLTC (green, dash-dotted), and BLDTT (blue, solid) with (a) M=2E7 targets, N=2E6 sources, (b) M=2E6 targets, N=2E7 sources, for MAC θ and interpolation degree n as above. For (a) $M > N$, the CP-BLTC outperforms the PC-BLTC, and for (b) $N > M$, the PC-BLTC outperforms the CP-BLTC except at low degree n, while the BLDTT outperforms both treecodes except in (b) for MAC $\theta = 0.9$ and low degree n, where the CP-BLTC runs slightly faster. Note that the CP-BLTC does not have an upward pass, and in the current GPU implementation of the BLDTT and PC-BLTC, the upward pass parallelizes less well than the downward pass, and with low degree n, it contributes a substantial fraction of the PC-BLTC and BLDTT run time. Nonetheless the results demonstrate the ability of the BLDTT to adapt to systems of unequal targets and sources.

5.6. Different kernels

Previous sections considered particles interacting through the Coulomb kernel, and here we demonstrate the kernel-independent nature of the BLDTT by applying it to three different kernels: (a) regularized Coulomb, $1/(r^2 + 0.005^2)^{1/2}$, (b) Yukawa, $\exp(-0.5r)/r$, (c) oscillatory, $\sin(\pi r)/r$. Fig. 10 shows the run time (s) versus relative ℓ_2 error for the BLDTT (blue, solid) and BLTC (red, dashed) with N=2E7 random uniformly distributed particles in the cube $[-1, 1]^3$ for MAC θ and interpolation degree n as above. The BLDTT has consistently better performance than the BLTC for all three kernels.

5.7. Memory usage

Next we consider the peak memory usage of the BLDTT and BLTC for 2E7 random uniformly distributed particles in $[-1, 1]^3$ interacting through the Coulomb kernel, where the peak memory is the maximum resident set size (MaxRSS) reported by the



(a) $1/(r^2+0.005^2)^{1/2}$ (b) $\exp(-0.5r)/r$ (c) $\sin(\pi r)/r$

Fig. 10. Different kernels, N=2E7 random uniformly distributed particles in $[-1,1]^3$, (a) regularized Coulomb, $1/(r^2+0.005^2)^{1/2}$, (b) Yukawa, $\exp(-0.5r)/r$, (c) oscillatory, $\sin(\pi r)/r$, run time (s) versus relative ℓ_2 error, BLTC (red, dashed), BLDTT (blue, solid), connected curves represent constant MAC θ (0.5 $\times$; 0.7 $\circ$; 0.9 $*$), interpolation degree $n=1,2,4,6,8,10$ increases from right to left on each curve, computed on one NVIDIA P100 GPU.

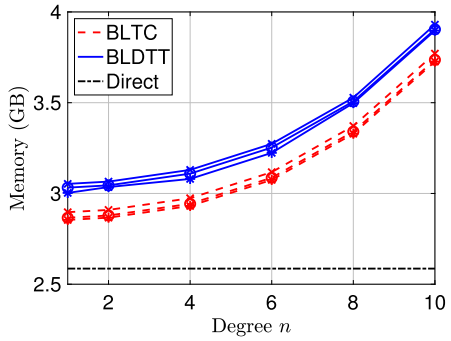


Fig. 11. Peak memory, N=2E7 random uniformly distributed particles in $[-1,1]^3$, peak memory (GB) versus interpolation degree n, BLDTT (blue, solid), BLTC (red, dashed), direct summation (black, dashed-dotted), connected curves represent constant MAC θ (0.5 $\times$; 0.7 $\circ$; 0.9 $*$), degree $n=1,2,4,6,8,10$ increases from right to left on each curve.

SLURM job scheduler. Fig. 11 shows the peak memory (GB) versus interpolation degree n for the BLDTT (blue, solid) and BLTC (red, dashed) with MAC θ and degree n as above. The peak memory for direct summation (black, dash-dotted) in this case is 2.586 GB. For a given MAC θ and degree n, the BLDTT uses about 5% more memory than the BLTC, which is due to storing the proxy particles and potentials in the target clusters (the BLDTT interaction lists use very little memory). For both the BLDTT and BLTC, smaller MAC leads to a slight increase in memory, with values within roughly 1% of each other for a given degree. At the highest degree $n=10$, the BLDTT uses about 50% more memory than direct summation, which is due to storing the proxy particles, charges, and potentials in the target and source clusters.

5.8. MPI strong scaling

We examine the MPI strong scaling of the BLDTT running on 1 to 32 NVIDIA P100 GPUs with one MPI rank per GPU. The particles are partitioned into compact domains by Trilinos Zoltan [66,67] and each domain resides on one GPU. Fig. 12 depicts the domain decomposition for N=1.6E6 random uniformly distributed particles in the cube $[-1,1]^3$ across (a) 8 ranks with 2E5 particles per rank, (b) 16 ranks with 1E5 particles per rank. Colors represent particles residing on different ranks.

Fig. 13 shows the run time versus the number of GPUs for systems with N=64E6 random particles having (a) uniform, (b) Gaussian, and (c) Plummer distributions as above. The computations use MAC $\theta=0.7$ and interpolation degree $n=8$ yielding error $\approx$ 1E–8. Results are shown for the BLDTT (blue) and BLTC (red), where the boxed numbers give the parallel efficiency and the dashed lines indicate ideal scaling. The BLDTT is 1.5–2.5$\times$ faster than the BLTC up to 32 GPUs, and the speedup improves for the non-uniform particle distributions. The BLDTT has slightly lower parallel efficiency than the BLTC except for the Plummer distribution with 32 GPUs. The parallel efficiency of the BLDTT remains 65% or higher up to 32 GPUs for all three distributions.

The lower parallel efficiency of the BLDTT compared to the BLTC can be explained by profiling the computations. Fig. 14 shows the component breakdown as a percent of run time for the (a) BLTC and (b) BLDTT uniform distribution results in Fig. 13(a). The components are the upward pass (blue), computing due to local sources and source clusters (orange), computing due to remote sources and source clusters (yellow), downward pass (purple), LET building and communication (green), and other (light blue) which includes tree building and interaction list building. The breakdown is based on timing results for the most expensive MPI rank in each case. Note that LET building accounts for a larger fraction of the run time as the number of ranks increases; this is due to the larger communication cost as more particles reside on remote ranks, and it is the primary factor that impedes ideal parallel scaling. The LET building time is nearly identical for the BLTC and BLDTT, but since the BLDTT runs faster, LET building accounts for a larger fraction of the run time, and this explains the lower parallel efficiency of the BLDTT compared to the BLTC in Fig. 13.

6. Conclusions

We presented a GPU-accelerated fast multipole method called BLDTT, which uses barycentric Lagrange interpolation for the near-field and far-field approximations [60,34] and dual tree traversal to construct the interaction lists [12,13]. The BLDTT builds adaptive trees of clusters on the target particles and source particles, where each cluster is the minimal bounding box of its particles, and a parent cluster may have 8, 4, or 2 children. The scheme replaces well-separated particle-particle interactions by adaptively chosen particle-cluster, cluster-particle, and cluster-cluster approximations given by barycentric Lagrange interpolation on a Chebyshev grid of proxy particles in each cluster. The BLDTT employs FMM-type upward and downward passes [4], although here they are adapted to interlevel polynomial interpolation. The BLDTT is kernel-independent and the approximations have a direct sum form that efficiently maps onto GPUs, where targets provide an outer level of parallelism and sources provide an inner level of parallelism. The code uses OpenACC directives for GPU acceleration, and the distributed memory parallelization builds locally essential trees on each rank [65], with MPI remote memory access for communication and recursive coordinate bisection for domain decomposition. The BLDTT code is available in the BaryTree library for fast summation of particle interactions [63].

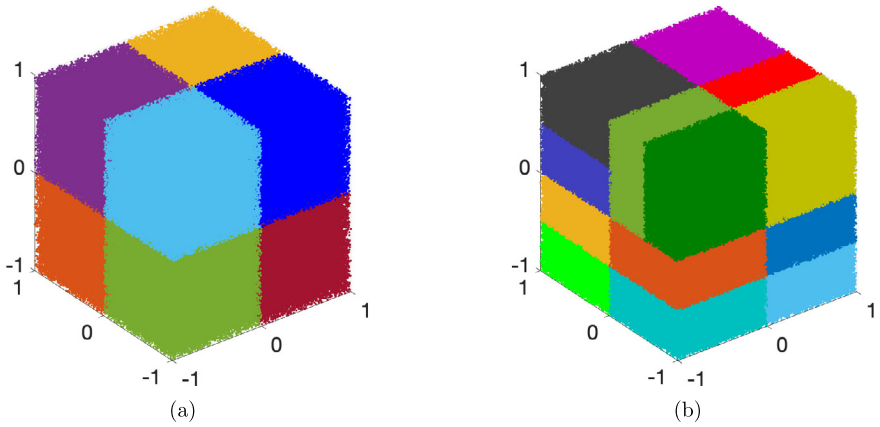


Fig. 12. Examples of domain decomposition, N=1.6E6 random uniformly distributed particles in $[-1,1]^3$, (a) 8 ranks with 2E5 particles per rank, (b) 16 ranks with 1E5 particles per rank, colors represent particles residing on different ranks, partitioning by Trilinos Zoltan [66,67].

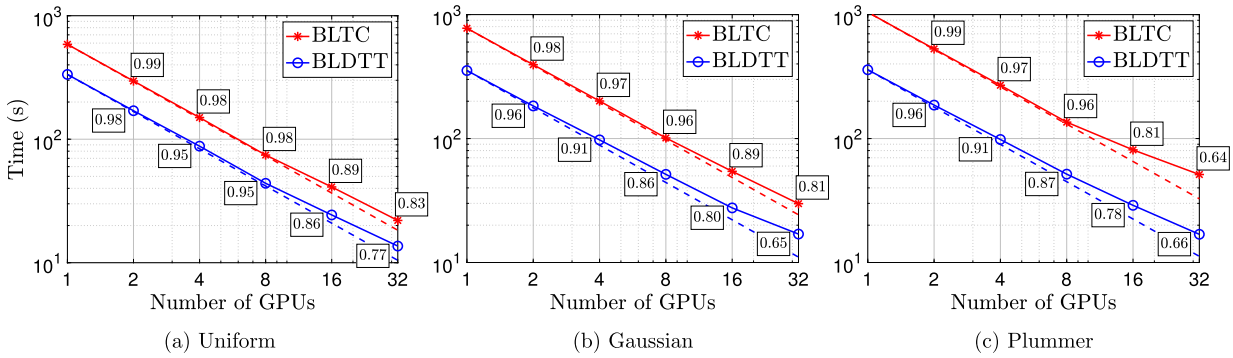


Fig. 13. MPI strong scaling, N=64E6 random particles in $[-1,1]^3$, (a) uniform, (b) Gaussian, (c) Plummer distributions, MAC $\theta = 0.7$, interpolation degree $n = 8$ yielding 7-8 digit accuracy, run time (s) versus number of GPUs, BLTC (red), BLDTT (blue), ideal scaling (dashed lines), parallel efficiency (boxed numbers).

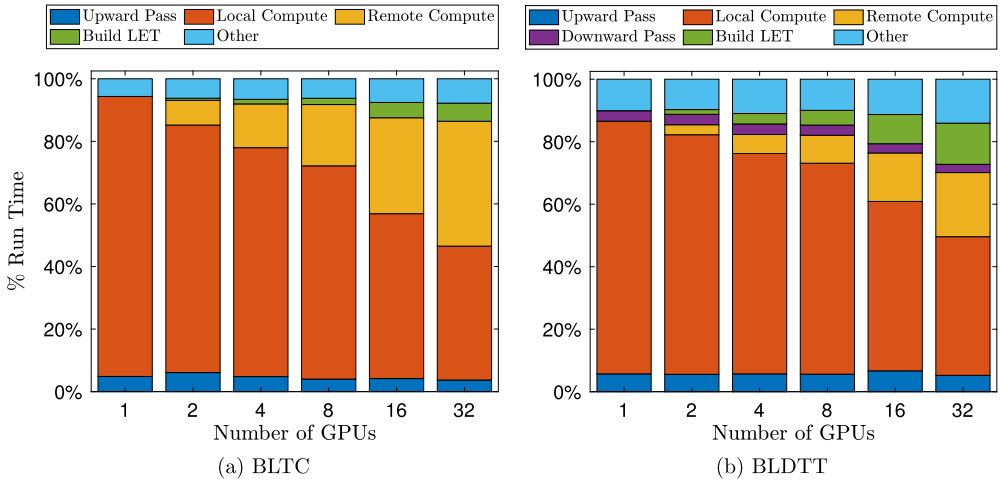


Fig. 14. Component breakdown of run time across 1 to 32 GPUs, 64E6 random uniformly distributed particles in $[-1,1]^3$, MAC $\theta = 0.7$, interpolation degree $n = 8$, error $\approx$ 1E-8, (a) BLTC, (b) BLDTT, upward pass (blue), computing due to local sources and source clusters (orange), computing due to remote sources and source clusters (yellow), downward pass (purple), LET building and communication (green), and other (light blue) including tree building and interaction list building, breakdown is based on timing results for most expensive MPI rank in each case.

The BLDTT was compared with the BLTC, an earlier particle-cluster barycentric Lagrange treecode [34]. For systems with N=1E5 to 1E8 particles running on one GPU, the BLTC scales like $O(N \log N)$ and the BLDTT scales like $O(N)$. Computations were presented for different random particle distributions (uniform, Gaussian, Plummer), domains (thin slab, square rod, spherical surface), and interaction kernels (singular and regularized Coulomb, Yukawa, oscillatory) and for unequal target and source particles. In terms of run time versus error, the BLDTT has consistently better performance than the BLTC. The BLDTT uses about 5% more memory than the BLTC, and with degree $n = 10$ it uses about 50% more memory than direct summation. We demonstrated the MPI strong scaling of the BLDTT and BLTC running on 1 to 32 GPUs for N=64E6 particles with 7–8 digit accuracy, where the BLDTT is 1.5–2.5$\times$ faster than the BLTC, and the parallel efficiency of the BLDTT is better than 65% up to 32 GPUs.

Future work to improve the efficiency of the BLDTT could investigate overlapping communication and computation, building tree nodes that span multiple ranks, using mixed-precision arithmetic, and employing barycentric Hermite interpolation [72]. We anticipate applying the BLDTT to speed up integral equation based implicit solvent computations [73] and density functional theory calculations [74].

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.

Acknowledgements

We thank the Referees for constructive comments that helped improve the manuscript. This work was supported by National Science Foundation grant DMS-1819094, Extreme Science and Engineering Discovery Environment (XSEDE) grants ACI-1548562, ASC-190062, and the Mcubed program and Michigan Institute for Computational Discovery and Engineering (MICDE) at the University of Michigan.

Appendix A. Details of upward pass

First we note a property of polynomial interpolation. Recall that given a function $f(x)$ and $n+1$ distinct points $s_k, k = 0:n$, the interpolating polynomial $p(x)$ can be written as

$$p(x) = \sum_{k=0}^{n} f(s_k) L_k(x), \tag{A.1}$$

where $L_k(x)$ are the Lagrange polynomials. If $f(x)$ itself is a polynomial of degree n, then

$$f(x) = \sum_{k=0}^{n} f(s_k) L_k(x), \tag{A.2}$$

because the left and right sides are both polynomials of degree n with the same values at the $n+1$ distinct points s_k.

Now we turn to the upward pass. Recall Eq. (9) for the definition of the proxy charges $\widehat{q}_k$ of a source cluster C_s,

$$\widehat{q}_k = \sum_{y_j \in C_s} L_k(y_j) q_j, \quad k = 0:n, \tag{A.3}$$

where y_j, q_j are the particles and charges of the cluster, and $L_k(y)$ are the Lagrange polynomials associated with the cluster. Suppose the cluster C_s has two child clusters $C_{s,i}$, $i = 1:2$ with interpolation points s_{k_i}, Lagrange polynomials $L_{k_i}(y)$, and proxy charges $\widehat{q}_{k_i}$. The parent proxy charge in Eq. (A.3) can be expressed in terms of the child proxy charges as follows,

$$\widehat{q}_k = \sum_{y_j \in C_s} L_k(y_j) q_j \tag{A.4a}$$

$$= \sum_{i=1}^{2} \sum_{y_j \in C_{s,i}} L_k(y_j) q_j \tag{A.4b}$$

$$= \sum_{i=1}^{2} \sum_{y_j \in C_{s,i}} \left(\sum_{k_i=0}^{n} L_k(s_{k_i}) L_{k_i}(y_j) \right) q_j \tag{A.4c}$$

$$= \sum_{i=1}^{2} \sum_{k_i=0}^{n} L_k(s_{k_i}) \sum_{y_j \in C_{s,i}} L_{k_i}(y_j) q_j \tag{A.4d}$$

$$= \sum_{i=1}^{2} \sum_{k_i=0}^{n} L_k(s_{k_i}) \widehat{q}_{k_i}. \tag{A.4e}$$

Step a is the definition of the parent proxy charges, step b splits this into the sum over the two child clusters, step c uses the relation in Eq. (A.2), step d rearranges the sums, and step e applies the definition of the child proxy charges. This result extends in a straightforward way to 3D, where a parent cluster may have eight, four, or two children. In summary, the upward pass computes the proxy charges of the leaf clusters by Eq. (A.4a) and then it ascends to the root of the source tree, computing the proxy charges of each parent source cluster from the child proxy charges by Eq. (A.4e). This is similar to the upward pass in the FMM, where the multipole moments of a parent cluster are computed from the moments of its children [4].

Appendix B. Details of downward pass

For simplicity we consider a target tree with two levels ($L = 2$). Recall Eq. (18),

$$\phi(x_i) \mathrel{+}= \sum_{m=1}^{2} \sum_{k_m=0}^{n} \phi(t_{k_m}^m) L_{k_m}^m(x_i), \tag{B.1}$$

where the inner sum interpolates the proxy potentials $\phi(t_{k_m}^m)$ at level m in the target tree directly to the target particle x_i, and $+=$ indicates that the results are combined with the potentials $\phi(x_i)$ previously computed in the DTT. The right side of Eq. (B.1) can be written as follows,

$$\sum_{m=1}^{2} \sum_{k_m=0}^{n} \phi(t_{k_m}^m) L_{k_m}^m(x_i)$$

$$= \sum_{k_1=0}^{n} \phi(t_{k_1}^1) L_{k_1}^1(x_i) + \sum_{k_2=0}^{n} \phi(t_{k_2}^2) L_{k_2}^2(x_i) \tag{B.2a}$$

$$= \sum_{k_1=0}^{n} \phi(t_{k_1}^1) L_{k_1}^1(x_i) + \sum_{k_2=0}^{n} \phi(t_{k_2}^2) \left(\sum_{k_1=0}^{n} L_{k_2}^2(t_{k_1}^1) L_{k_1}^1(x_i) \right) \tag{B.2b}$$

$$= \sum_{k_1=0}^{n} \left(\phi(t_{k_1}^1) + \sum_{k_2=0}^{n} \phi(t_{k_2}^2) L_{k_2}^2(t_{k_1}^1) \right) L_{k_1}^1(x_i). \tag{B.2c}$$

Step a splits the sum over the two levels, step b uses the relation in Eq. (A.2), and step c rearranges the sums. This yields an alternative version of Eq. (B.1),

$$\phi(x_i) \mathrel{+}= \sum_{k_1=0}^{n} \left(\phi(t_{k_1}^1) + \sum_{k_2=0}^{n} \phi(t_{k_2}^2) L_{k_2}^2(t_{k_1}^1) \right) L_{k_1}^1(x_i), \tag{B.3}$$

where instead of interpolating from $t_{k_1}^1$ to x_i *and* from $t_{k_2}^2$ to x_i (step a), one interpolates from $t_{k_2}^2$ to $t_{k_1}^1$, combines with previously computed results at $t_{k_1}^1$, and interpolates from $t_{k_1}^1$ to x_i (step c). In this expression, the first term in parentheses corresponds to Eq. (20) and the second term corresponds to Eq. (19). The procedure generalizes to trees of any depth. This is similar to the downward pass in the FMM, which computes potentials using multipole-to-local and local-to-local translations of expansion coefficients [4].

References

[1] R.W. Hockney, J.W. Eastwood, Computer Simulation Using Particles, Taylor & Francis, 1988.
[2] U. Essmann, L. Perera, M. Berkowitz, T. Darden, H. Lee, L. Pederson, J. Chem. Phys. 103 (1995) 8577–8593.
[3] J.E. Barnes, P. Hut, Nature 324 (1986) 446–449.
[4] L. Greengard, V. Rokhlin, J. Comput. Phys. 73 (1987) 325–348.
[5] W. Hackbusch, Z.P. Nowak, Numer. Math. 54 (1989) 463–491.
[6] W. Hackbusch, Hierarchical Matrices: Algorithms and Analysis, Springer, 2015.
[7] D.J. Hardy, Z. Wu, J.C. Phillips, J.E. Stone, R.D. Skeel, K. Schulten, J. Chem. Theory Comput. 11 (2015) 766–779.
[8] K. Lindsay, R. Krasny, J. Comput. Phys. 172 (2) (2001) 879–907.
[9] H. Cheng, L. Greengard, V. Rokhlin, J. Comput. Phys. 155 (2) (1999) 468–498.
[10] Y. Andoh, N. Yoshii, K. Fujimoto, K. Mizutani, H. Kojima, A. Yamada, S. Okazaki, K. Kawaguchi, H. Nagao, K. Iwahashi, F. Mizutani, K. Minami, S.-i. Ichikawa, H. Komatsu, S. Ishizuki, Y. Takeda, M. Fukushima, J. Chem. Theory Comput. 9 (7) (2013) 3201–3209.
[11] N.A. Gumerov, R. Duraiswami, J. Comput. Phys. 227 (18) (2008) 8290–8313.
[12] A.W. Appel, SIAM J. Sci. Stat. Comput. 6 (1) (1985) 85–103.
[13] W. Dehnen, J. Comput. Phys. 179 (2002) 27–42.
[14] K. Esselink, Inf. Process. Lett. 41 (3) (1992) 141–147.
[15] M.S. Warren, J.K. Salmon, Comput. Phys. Commun. 87 (1–2) (1995) 266–290.
[16] S.-H. Teng, SIAM J. Sci. Comput. 19 (2) (1998) 635–656.
[17] K. Lorenzen, M. Schwörer, P. Tröster, S. Mates, P. Tavan, J. Chem. Theory Comput. 8 (10) (2012) 3628–3636.
[18] J. Coles, M. Masella, J. Chem. Phys. 142 (2) (2015) 024109.
[19] K. Taura, J. Nakashima, R. Yokota, N. Maruyama, in: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, 2012, pp. 617–625.
[20] R. Yokota, J. Algorithms Comput. Technol. 7 (3) (2013) 301–324.
[21] W. Dehnen, Comput. Astrophys. Cosmol. 1 (1) (2014) 1–23.
[22] B. Lange, P. Fortin, in: Proc. 20th Int. Eur. Conf. Parallel Distrib. Comput. (Euro-Par 2014), 2014, pp. 716–727.
[23] L. Greengard, J. Huang, J. Comput. Phys. 180 (2) (2002) 642–658.
[24] Z.-H. Duan, R. Krasny, J. Comput. Chem. 22 (2) (2001) 184–195.
[25] B. Shanker, H. Huang, J. Comput. Phys. 226 (2007) 732–753.
[26] P. Li, H. Johnston, R. Krasny, J. Comput. Phys. 228 (2009) 3858–3868.
[27] C.R. Anderson, SIAM J. Sci. Stat. Comput. 13 (1992) 923–947.
[28] J. Makino, J. Comput. Phys. 151 (1999) 910–920.
[29] L. Ying, G. Biros, D. Zorin, J. Comput. Phys. 196 (2) (2004) 591–626.
[30] G. Schmidlin, C. Lage, C. Schwab, Eng. Anal. Bound. Elem. 27 (2003) 469–490.
[31] S. Börm, M. Löhndorf, J.M. Melenk, Numer. Math. 99 (2005) 605–643.
[32] W. Fong, E. Darve, J. Comput. Phys. 228 (23) (2009) 8712–8725.
[33] J.R. Saverin, D. Marten, C. Nayeri, G. Pechlivanoglou, C.O. Paschereit, in: AIAA SciTech Forum: 2018 Wind Energy Symposium, Kissimmee, Florida, AIAA 2018-0254, 2018, pp. 1–25.
[34] L. Wang, R. Krasny, S. Tlupova, Commun. Comput. Phys. 28 (2020) 1415–1436.
[35] L. Cambier, E. Darve, SIAM J. Sci. Comput. 41 (3) (2019) A1652–A1680.
[36] X. Xing, E. Chow, SIAM J. Matrix Anal. Appl. 41 (1) (2020) 221–243.
[37] L. Ying, G. Biros, D. Zorin, H. Langston, in: Proc. 2003 ACM/IEEE Conf. Supercomput. (SC03), 2003, p. 14.
[38] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, G. Biros, Commun. ACM 55 (2012) 101–109.
[39] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, T. Takahashi, SIAM J. Sci. Comput. 36 (1) (2014) C66–C93.
[40] W.B. March, B. Xiao, S. Tharakan, C.D. Yu, G. Biros, in: Proc. Int. Conf. High Perf. Comput. Networking, Storage Anal (SC15), 2015, pp. 24:1–24:12.
[41] D. Malhotra, G. Biros, Commun. Comput. Phys. 18 (3) (2015) 808–830.
[42] D. Malhotra, G. Biros, ACM Trans. Math. Softw. 43 (2016) 1–27.
[43] H. Huang, X. Xing, E. Chow, ACM Trans. Math. Softw. (2020).
[44] E. Elsen, M. Houston, V. Vishal, E. Darve, P. Hanrahan, V. Pande, in: Proc. 2006 ACM/IEEE Conf. Supercomput. (SC06), 2006, p. 188.
[45] L. Nyland, M. Harris, J. Prins, Fast *N*-Body Simulation with CUDA, GPU Gems, vol. 3, 2009, pp. 677–695.
[46] W. Geng, F. Jacob, Comput. Phys. Commun. 184 (2013) 1490–1496.
[47] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, M. Taiji, in: Proc. Int. Conf. High Perf. Comput. Networking, Storage Anal. (SC09), 2009, pp. 1–12.
[48] M. Burtscher, K. Pingali, in: W.-M. Hwu (Ed.), GPU Computing Gems Emerald Edition, Morgan Kaufmann/Elsevier, 2011, pp. 75–92, Ch. 6.
[49] J. Bédorf, E. Gaburov, S.P. Zwart, J. Comput. Phys. 231 (7) (2012) 2825–2839.
[50] J. Bédorf, E. Gaburov, M.S. Fujii, K. Nitadori, T. Ishiyama, S.P. Zwart, in: Proc. Int. Conf. High Perf. Comput. Networking, Storage Anal (SC14), 2014, pp. 54–65.
[51] G. Lukat, R. Banerjee, New Astron. 45 (2016) 14–28.
[52] N. Vaughn, L. Wilson, R. Krasny, in: Proc. 21st IEEE Int. Workshop Parallel Distrib. Sci. Eng. Comput. (PDSEC 2020), 2020, pp. 701–710.
[53] R. Yokota, J.P. Bardhan, M.G. Knepley, L.A. Barba, T. Hamada, Comput. Phys. Commun. 182 (2011) 1272–1283.
[54] T. Takahashi, C. Cecka, E. Darve, Parallel Comput. (InPar) (2012) 1–14.
[55] W. Boukaram, G. Turkiyyah, D. Keyes, ACM Trans. Math. Softw. 45 (1) (2019) 3:1–3:28.
[56] W. Boukaram, G. Turkiyyah, D. Keyes, SIAM J. Sci. Comput. 41 (2019) C339–C366.
[57] B. Kohnke, C. Kutzner, H. Grubmüller, J. Chem. Theory Comput. 16 (2020) 6938–6949.
[58] R. Yokota, L.A. Barba, Comput. Sci. Eng. 14 (3) (2012) 30–39.
[59] P. Fortin, M. Touche, Int. J. High Perform. Comput. Appl. (2019) 1–13.
[60] J.-P. Berrut, L.N. Trefethen, SIAM Rev. 46 (3) (2004) 501–517.
[61] H.A. Boateng, Cartesian treecode algorithms for electrostatic interactions in molecular dynamics simulations, Ph.D. thesis, University of Michigan, 2010.
[62] H.A. Boateng, R. Krasny, J. Comput. Chem. 34 (25) (2013) 2159–2167.
[63] N. Vaughn, L. Wilson, R. Krasny, BaryTree, https://github.com/Treecodes/BaryTree, 2020.
[64] H.E. Salzer, Comput. J. 15 (1972) 156–159.
[65] M.S. Warren, J.K. Salmon, in: Proc. 1992 ACM/IEEE Conf. Supercomput. (SC92), 1992, pp. 570–576.
[66] The Zoltan Project Team, The Zoltan Project, https://trilinos.github.io/zoltan.html, 2020.
[67] E.G. Boman, U.V. Catalyurek, C. Chevalier, K.D. Devine, Sci. Program. 20 (2) (2012) 129–150.
[68] J.R. Hammond, J. Dinan, P. Balaji, I. Kabadshow, S. Potluri, V. Tipparaju, in: 6th Conf. Partitioned Global Address Space Programming Models, PGAS 12, 2012.
[69] H.C. Plummer, Mon. Not. R. Astron. Soc. 71 (1911) 460–470.
[70] H. Dejonghe, Mon. Not. R. Astron. Soc. 224 (1987) 13–39.
[71] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G.D. Peterson, R. Roskies, J.R. Scott, N. Wilkins-Diehr, Comput. Sci. Eng. 16 (5) (2014) 62–74.
[72] R. Krasny, L. Wang, Comput. Math. Biophys. 7 (2019) 73–84.
[73] W. Geng, R. Krasny, J. Comput. Phys. 247 (2013) 62–78.
[74] N. Vaughn, V. Gavini, R. Krasny, J. Comput. Phys. 430 (2021) 110101.