Communication Reducing Algorithms for Distributed Hierarchical N-Body Problems with Boundary Distributions

Mustafa Abduljabbar[1](✉), George S. Markomanolis[2], Huda Ibeid[1], Rio Yokota[3], and David Keyes[1]

[1] Extreme Computing Research Center (ECRC),
King Abdullah University of Science and Technology (KAUST),
Thuwal, Saudi Arabia
mustafa.abduljabbar@kaust.edu.sa
[2] KAUST Supercomputing Laboratory (KSL),
King Abdullah University of Science and Technology (KAUST),
Thuwal, Saudi Arabia
[3] Global Scientific Information and Computing Center (GSIC),
Tokyo Institute of Technology (TITECH), Tokyo, Japan

Abstract. Reduction of communication and efficient partitioning are key issues for achieving scalability in hierarchical N-Body algorithms like Fast Multipole Method (`FMM`). In the present work, we propose three independent strategies to improve partitioning and reduce communication. First, we show that the conventional wisdom of using space-filling curve partitioning may not work well for boundary integral problems, which constitute a significant portion of `FMM`'s application user base. We propose an alternative method that modifies orthogonal recursive bisection to relieve the cell-partition misalignment that has kept it from scaling previously. Secondly, we optimize the granularity of communication to find the optimal balance between a bulk-synchronous collective communication of the local essential tree and an RDMA per task per cell. Finally, we take the dynamic sparse data exchange proposed by Hoefler et al. [1] and extend it to a *hierarchical* sparse data exchange, which is demonstrated at scale to be faster than the MPI library's `MPI_Alltoallv` that is commonly used.

Keywords: N-body methods · Fast multipole method · Load balancing · Communication reduction

1 Introduction

The N-body problem is a kernel in many scientific simulations in which the behavior of the system is defined from mutual interactions between discrete entities (e.g., molecules, charges, astrophysical bodies). The N-body algorithm sums up contributions due to all particles in the system, which results in quadratic complexity. The Barnes-Hut treecode, which subdivides the 2D/3D domain into

J.M. Kunkel et al. (Eds.): ISC High Performance 2017, LNCS 10266, pp. 79–96, 2017.
DOI: 10.1007/978-3-319-58667-0_5

quad/octrees, brings the complexity down to $\mathcal{O}(N \log N)$ by hierarchically clustering the sources into multipole expansions. FMM clusters the targets into local expansions to bring the complexity further down to $\mathcal{O}(N)$. For mathematical foundations of the multipole expansions, see [2–4]. Among the applications of FMM are [5,6] where protein-protein encounter within a biomolecular dynamics solver is accelerated by using FMM to solve the boundary integral equation, which is used to discretize the linearized Possion-Boltzmann equation. In [7] all-atom molecular dynamics is performed to simulate the conditions of living cells by calculating energy at target proteins in a solvent and a molecular crowder using FMM. It is also used to speedup the matrix-vector multiplication, which arises from electromagnetic scattering problems [8]. Other applications include gravity simulations [9,10].

Due to their increased importance in large-scale simulations, there is now a considerable literature on implementing parallel hierarchical N-body solvers, e.g., FMM. Also, since they are among Berkeley's seven dwarfs, the numerical methods that are believed to be the most impactful in science and engineering according to [11], it is important to address issues arising at exascale especially the increasing cost of data movement (through memory hierarchy or network) as opposed to floating point operations. Even though many of the current FMM implementations are scalable to the full machine they run on, a communication reducing approach that works on at least an order of magnitude more nodes tends to be rarely the emphasis of these implementations. This tendency is justified in accordance to the trend in enhancing a node with multi/many-core capabilities. However, even within a many-core node, more sophisticated methods should be used to place and exchange data to get the maximum performance reported by the vendor. This is already implied in equipping the second generation of Intel® Xeon Phi™ processors code-named Knights Landing (KNL) with memory 'clustering modes'. Therefore, ideas presented in this paper complement the literature although they mainly target distributed memory.

An example work that achieves full machine scalability using GPUs is that of Bédorf et al. [9], where a parallel algorithm for sparse tree construction and traversal that works completely on the GPU is introduced. At the construction phase, they map the 3D coordinates to Hilbert's linear (n-bit) addresses, then particles are sorted to achieve locality in memory. To avoid the typical sequential insertions to build Hilbert trees [12], one particle is assigned per GPU thread. A level-wise mask is applied successively on each particle to discover its predecessors such that cells with less than N_{leaf} are considered leaves. Grouping of particles is done using parallel compact algorithm. To exploit the massively parallel GPU threads, a breadth-first traversal is used to carry out the computation. They report a processing rate of 2.8 million particles per second. This work was extended to an MPI parallel version where 24.77 PFlop/s (mixed precision) on the full Titan system [13] was achieved.

Speck et al. [14] report scalability on up to 262,144 cores by introducing temporal parallelism (parallel-in-time algorithm) on top of MPI/Pthreads spatial decomposition to overcome the strong scaling limits when the number of particles per node becomes too small. The scalability is shown for up to 4M particles; then when they take advantage of shared and distributed memory parallelism, and exploit the overlap of data-exchange and computation, they calculate 2 billion particles on 262,144 cores of JUGENE, according to [15]. Lashuk et al. [16] propose an FMM implementation that scales on up to 196,608 cores by providing a novel domain-specific bulk synchronous all-reduce algorithm for remote tree communication. They report communication complexity of $O(\sqrt{P} * (\frac{N}{P})^{2/3})$, which comes from their hypercube alltoall communication scheme. Hoefler et al. [1] discuss the time and memory complexity of the common protocols used for the dynamic sparse data exchange problem and develop the non-blocking exchange protocol ($\mathcal{NBX}$) with constant memory overhead. Their novel algorithm improves the runtime of sparse data-exchange up to 8,192 processors of Bluegene/P by a factor of 5.6. They prove and model a generic time complexity of $\mathrm{O}(\log P)$ using the LogGP model.

Zandifar et al. [17] provide a parallel FMM implementation as a benchmark for their high-level skeletons (abstract parallel patterns) framework which executes on top of the STAPL runtime system that dynamically schedules task on highly heterogeneous architectures. They reuse several parallel patterns like the bucket-sort and alltoall to perform geometric bisection and to aggregate the local essential tree (LET) respectively. They achieve comparable performance to the corresponding base MPI implementation by taking advantage of the underlying data-driven execution and asynchronous task scheduling guaranteed by the runtime system. Many features of Charm++ like task migration and Structured Control Flow are augmented in [18] to overlap computation with the communication of the local essential tree (LET).

Contributions of the present work can be summarized as follows:

- A novel demonstration that shows a weakness in Hilbert's space-filling interval partitioning for boundary element distributions.
- A communication scheme with adjustable granularity, which enables the overlap of local essential tree communication with computation that otherwise cannot be overlapped.
- Introduction of the adaptive hierarchical sparse data exchange ($\mathcal{HSDX}$), a neighborhood collective communication algorithm for exchanging the global tree in a few steps by direct near-field communication only.

In Sect. 2, we describe our adopted partitioning techniques and justify our choice in detail. Section 4 describes different communication strategies that we adopt in order to avoid bulk synchronous LET communication. We also describe the adopted load-balancing strategies and the communication complexity analysis of our approach. Finally, we demonstrate our scalability and evaluation results.

2 Partitioning Schemes for the Fast Multipole Method (FMM)

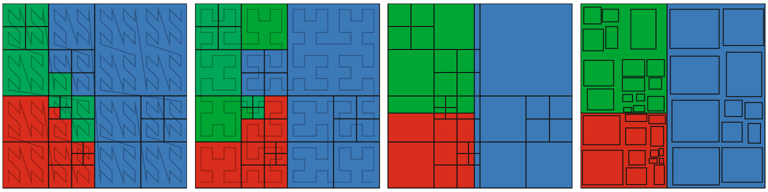
There are two traditional objectives associated with good partitioning of the N-body problem: evenly splitting data among partitions to achieve work balance, and providing efficient access to non-local data. There is no optimal approach that can simultaneously handle these two objectives, because of strict considerations on locality of data for high arithmetic intensity, granularity, and the size of communication, which can vary based on space-time proximity of partitions.

2.1 Preliminaries

Partitioning schemes for fast N-body methods can be categorized into orthogonal recursive bisections (ORB) [19] or hashed octrees (HOT) [20].

Orthogonal Recursive Bisection (ORB). The ORB [19] forms a balanced binary tree by finding a geometric bisector that splits the number of particles equally at every bisection of the tree. The direction of the geometric bisector alternates orthogonally $(x, y, z, x, \ldots)$ to form a cascade of rectangular subdomains that contain equal number of particles similar to Fig. 1(c). For nonuniform distributions the aspect ratio of the subdomain could become large, which leads to suboptimal interaction list size and communication load. This problem can be alleviated by choosing the direction of the geometric bisector to always split in the longest dimension. The original method is limited to cases where the number of processes is a power of two, but the method can be extended to non-powers-of-two by using multi sections instead of bisections [21].

Hashed Oct-Tree (HOT). In HOT, initially proposed by [20], the domain is partitioned by splitting Morton/Hilbert ordered space filling curves into equal segments as shown in Fig. 1(a) and (b). Morton/Hilbert ordering maps the geometrical location of each particle to a single key. The value of the key depends on the depth of the tree at which the space filling curve is drawn. Three bits of the key are used to indicate which octant the particle belongs to at every level of the octree. Therefore, a 32-bit unsigned integer can represent a tree with 10 levels, and a 64-bit unsigned integer can represent a tree with 21 levels. Directly mapping this key to the memory address is inefficient for non-uniform distributions since most of the keys will not be used. Therefore, a hashing function is used to map the Morton/Hilbert key to the memory address of particles/cells.



(a) HOT (Morton) (b) HOT (Hilbert) (c) ORB (d) Present method

Fig. 1. Schematic of different partitioning schemes. (a) Shows the hashed octree with Morton keys. (b) Shows the hashed octree with Hilbert keys. (c) Shows the orthogonal recursive bisection with an underlying global tree. (d) Is the present method using an orthogonal recursive bisection with independent local trees and tight bounding boxes.

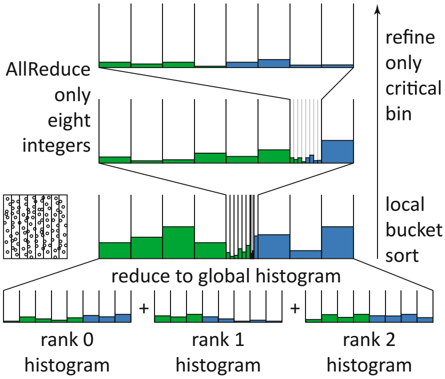
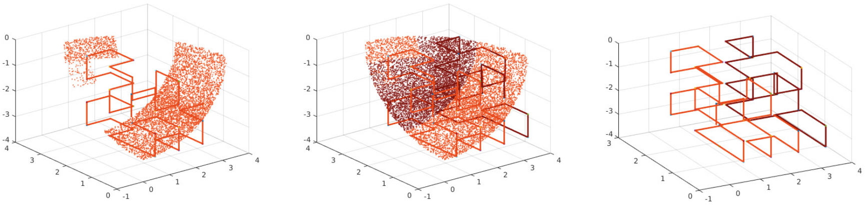


Fig. 2. Histogram-based partitioning scheme.

2.2 Adopted Partitioning Strategies

Parallel Sampling-Based Techniques for Finding Splitters/Bisectors. Parallel sampling-based techniques have proven to be useful for both finding the bisectors in ORB [21] and finding the splitting keys in HOT [22]. Both ORB and HOT are constructing parallel tree structures, but in different ways. There is an analogy between parallel tree construction and parallel sorting. The idea behind ORB is analogous to merge sort, where a divide and conquer approach is taken. HOT is analogous to radix sort, where each bit of the key is examined at each step. Therefore, sampling-based techniques that are known to be effective for parallel sorting are also effective for parallel tree partitioning. The partitioning can be separated into two steps. The first step is to find the bisectors/key-splitters by using a sampling-based parallel sorting algorithm. An example of such sampling-based partitioning is shown in Fig. 2. Sorting is only performed among the buckets (not within them) and this is done only locally. The only global information that is communicated is the histogram counts, which is only a few integers and can be done efficiently with an `MPI_allreduce` operation. The bins can be iteratively refined to narrow the search for the splitter of the HOT key or ORB bisector. This will determine the destination process for each particle. The second step is to perform an all-to-all communication of the particles. Since the ORB bisector is one floating point number and the HOT key is one integer, it is much less data than sending around particle data at each step of the parallel sort.

Weakness in Space-Filling Partitioning for Boundary Distributions. It is well-known that the main advantage of Hilbert curve as opposed to Morton is its locality preserving properties in 2D. It is not clear, however, to what extent we can generalize this property in higher dimensions [23]. As a counterexample to the locality property, we observe that it is not entirely preserved in case of 3D boundary element distributions, which increases the distributed interaction list size. The reason for that comes from the intuitive notion of space-filling



(a) Hilbert partition A.

(b) A, B combined.

(c) Underlying Hilbert curve.

Fig. 3. A Hilbert partition from a boundary spherical distribution viewed from different perspectives. A space discontinuity exists even though partitions are in correct Hilbert order due to the existence of hollow space in orthogonal dimensions.

curves, that is, when the space is not filled, e.g., in boundary spherical distribution, interpolation of spatial points to Hilbert curve does not necessarily map to keys that are continuous in space. This is attributable to the fact that keys are not interpolated in their natural order, since points are spread out on surface patches. Figure 3 shows particles laid out in their respective Hilbert order. Due to the geometry of the space-filling curve, movement across dimensions happens orthogonally, hence, if hollow space is encountered in the orthogonal direction, it will introduce discontinuity in the partition as in Fig. 3(a). Clearly, this does not apply to uniform dense distributions, which comprise many classical applications of `FMM`, making HOT partitioning an optimal choice in such cases.

Hybrid Partitioning. In our implementation, we choose a modified version of ORB over HOT for a few other reasons. One of the main reasons is that we were able to improve a major defect of ORB – partition-cell alignment issue. Since geometrically closer points interact more densely with each other, it is crucial to keep the particles in the same cell on the same process in order to minimize communication. However, if a global Morton/Hilbert key is used to construct the local trees, the ORB may place a bisector in the middle of a cell as shown in Fig. 1(c). This results in an increase in the interaction list size. We avoid this problem by using local Morton/Hilbert keys that use the bounds of the local partition. This may at first seem to increase the interaction list near the partition boundaries since two misaligned tree structures are formed. However, when one considers the fact that the present method squeezes the bounding box of each cell to tightly fit the particles as shown in Fig. 1(d), it can be seen that the cells are not aligned at all in the first place. Furthermore, our flexible definition of the multipole acceptance criteria optimizes the interaction list length for a given accuracy regardless of the misalignment.

3 Communication of the Local Essential Tree

Once particles are partitioned, those in the local domain are used to construct a local tree. We use a completely local construction of the octree using the local bounding box, instead of using a global Morton/Hilbert key that is derived from the global bounding box. This allows us to reuse all parts of the serial code and only add a few routines for the partitioning, grafting of trees, and communication. Therefore, any modification in the serial code is immediately reflected in the parallel code.

After the local tree structure is constructed, a post-order traversal is performed on the tree structure and Particle-to-Multipole (P2M) and Multipole-to-Multipole (M2M) kernels are executed bottom up. The P2M kernel is executed only at the leaf cells. It loops over all particles in the leaf cell to form the multipole expansion at the center of the leaf cell. The M2M kernel is executed only for the non-leaf cells. It loops over all child cells and translates the multipole expansions from its children's centers to its center.

Once the multipole expansions for all local cells have been determined, the multipole expansions are sent to the necessary processes in a sender-initiated fashion [24]. This reduces the latency by communicating only once, rather than sending a request to remote processes and then receiving the data. Such sender-initiated communication schemes were common in cosmological N-body codes since they tend to use only monopoles, and in this case the integer to store the requests is as large as the data itself if they were to use a request-based scheme. This data is used to construct the local essential tree (LET), that is, the union of all trees representing the entire domain as seen by the local process [25]. It gets coarser depending on the distance of the remote cell. In the present method, it is formed by simply grafting the root nodes of the remote trees. In conventional parallel FMM codes, a global octree is formed and partitioned using either HOT or ORB. Therefore, the tree structure was severed in many places, which caused the merging of the LET to become quite complicated. Typically, code for merging the LET would take a large portion of a parallel FMM code, and this made it difficult to implement new features such as periodic boundary conditions, mutual interaction, more efficient translation stencils, and dual tree traversals. ExaFMM[1] is able to incorporate all these extended features and still maintain a fast pace of development because of this simplification in how the global tree structure is geometrically separated from the local tree structure.

While the remote information for the LET is being transferred, the local tree can be traversed. Conventional fast N-body methods overlap the entire LET communication with the entire local tree traversal. The LET communication becomes a bulk-synchronous MPI_alltoallv type communication, where processes corresponding to geometrically far partitions send logarithmically less information, thus resulting in $O(\log P)$ communication complexity where P is the number of processes. Nonetheless, in traditional fast N-body codes this part is performed in a bulk-synchronous manner.

[1] ExaFMM is an open-source code base to utilize fast multipole algorithms, in parallel, and with GPU capability. Algorithms pertaining to partitioning and communication reduction are all available on the public repository https://github.com/exafmm/exafmm.

4 Communication Reduction for the Adaptive Tree

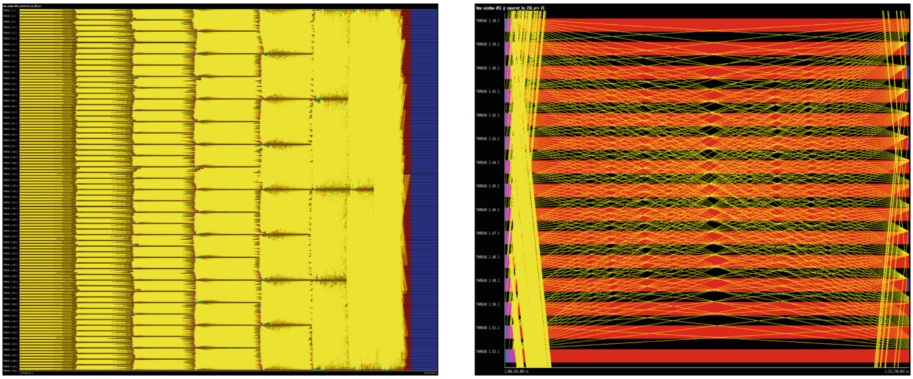
In the following sections, we present different novel techniques that can be used to do the hierarchically sparse data exchange ($\mathcal{HSDX}$) of the adaptive FMM tree, which are generally applicable to a variety of algorithms constituting Definition 1. The optimization of global tree communication is essential to achieve strong scaling especially at a large scale. Such class of communication becomes very challenging due to the fact that ExaFMM has a highly optimized serial code that utilizes many-core parallelism, making communication dominates even when overlapped with computation. The natural solution to this problem is to strong scale communication, but to our knowledge, it is not straightforward to achieve this for practical reasons such as network congestion, growing interaction lists, and the different implementations of some MPI collectives that do not scale by definition e.g., MPI_Alltoallv. Therefore, it is important to look at these caveats while implementing a domain-specific communication scheme of the global FMM tree.

Definition 1. *Let $\mathcal{T}$ be a global adaptive tree with $\mathcal{L}$ levels numbered from $l_0 - l_k$ (coarse to fine) and partitioned to $\mathcal{P}$ processes. s is the "essential" subtree size such that $0 < s < \mathcal{S}$. $P_i, P_j \subset l_k$, if the finest level P_i, P_j share is k. We have a hierarchically sparse data exchange $\mathcal{HSDX}$ if for $P_i, P_j \subset l_1$ and $P_i, P_v \subset l_2$, $s_1 < s_2$ and $s_1! = 0$*

4.1 Overlapping Computation Depending on Communication Granularity

Asynchronous communication is a limiting factor to performance at exascale especially when done collectively. This appears to be the case for hierarchical algorithms such as FMM and Multigrid method (MG). Hence, communication needs to be balanced and efficiently overlapped with local work. In FMM, it is known that a substantial amount of time is spent in doing local Multipole-to-Local (M2L) and Particle-to-Particle (P2P) computations, but the question is how often we need to communicate to reduce blocking for data given the problem size, distribution and scale. To answer this question, we have parametrized our FMM to accept different granularities of communication represented by the size of the LET's subset. The subsets may contain non-leaf cells requiring $O(p)$ steps for $p =$ order of multipole expansion (higher p increases arithmetic intensity for low-level kernels) or leaf cells requiring $O(N/P)^2$ steps for $P =$ number of processes. The typical case would be to call a blocking MPI_Recv on the expected tag because there is no useful work to do in the current context; however, since MPI does not provide guarantees on the order of messages when used in mixed mode, our code consumes the available subtree and marks it as "traversed". This mechanism will maximize concurrency and minimize the message queuing time. The calling task will keep traversing until requested cell is received or traversed by another task.

Conventional parallel N-body methods use a bulk-synchronous MPI_alltoallv to communicate the whole LET at once, and overlap this communication with the local tree traversal to hide latency. One could over-decompose the



(a) Hypercube communication of 256 MPI processes. Interactions get coarser as we move right and finish in LogP steps.

(b) A zoomed-in version of the flat communication of 256 MPI processes, potentially causing contention when scaling on a supercomputer's network.

Fig. 4. Implemented communication patterns in `FMM` as visualized by Extrae. (Color figure online)

LET down to a per cell request, and then aggregate the communication to the optimal granularity. The bulk-synchronous communication model can be thought of as an extreme case of aggregation, while something like an RDMA per task per cell would be at the other end of the granularity spectrum. There is a caveat: We still require further tuning to reduce global communication by indirectly relaying multipoles through neighbor processes, as we will show in Sect. 4.2 using Algorithm 1.

4.2 Hierarchical Sparse Data Exchange Protocol ($\mathcal{HSDX}$)

Lashuk et al. [26] define a set of parameters that denote the interaction lists, i.e., U-, V-, W- and X-lists of the `FMM` tree. The same analogy can be used for describing the relationship between adjacent processes such that exchanging the entire LET can happen in a few steps. The mentioned lists constitute the adjacent nodes/processes through which global cells that contribute to the local tree are relayed. For the majority of the spatial N-Body partitioning methods, we can use the subdomain's bounding box to depict partitions that share a face, an edge or a vertex in $O(1)$ steps using Lemma 1. This enables us to create a breadth-first data exchange graph that starts from the local tree and covers all the cells from the essential tree. Each node in the graph contains the corresponding partition id and the adjacent partition id, which is needed since communication strictly happens between adjacent nodes. Figure 5 shows the exchanges needed to receive the entire LET by target process (3,3), with overlapping direct clusters enclosed in dashed squares. The corresponding data exchange graph of node (3,3) contains a node with id (1,5) and an adjacent id of (2,4), meaning that cell data of (1,5) can be acquired through (2,4) in the second stage of exchanges. To inherently achieve algorithmic balance, we hardwire edges in such a way that

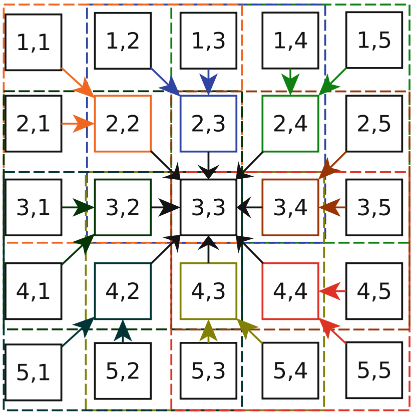


Fig. 5. The underlying data exchange graph of central process (3,3) within a uniform 2D grid of processes.

messages are evenly distributed over direct neighbors. If we start with direct neighbor (2,4), a naive approach would exhaust all its direct neighbors, namely {(1,3), (1,4), (1,5), (2,5), (3,5)}, thus overloads its buffers and causes imbalance. The next neighbor (3,4) will only have (4,5) data to relay. Therefore, we design our communication graph such that for internal processes in a uniform domain, the average number of messages received from direct neighbors in each step is $\left\lceil \frac{5^D-3^D}{3^D-1} \right\rceil$. Using notations from Table 2, we can generalize this formula to non-uniform domains if we turn it into

$$NB = \left\lceil \frac{\tau(P,1) - \zeta(\Omega(P))}{\zeta(\Omega(P)) - 1} \right\rceil \qquad (1)$$

We finally reach a stage where each process has access to the near and far-field interactions, thus accomplishing global communication using multiple calls to `MPI_Neighbor_alltoallv`. Algorithm 1 and Table 2 summarize our method.

Time Complexity of the adaptive $\mathcal{HSDX}$. A good lower bound complexity for $\mathcal{HSDX}$ is $\mathcal{NBX}$ i.e. $\Omega(\log P)$ from [1], when non-neighbor data exchange is extremely sparse or non-existent. The hierarchical sparsity in Definition 1 increases as we move away from target processes. The data exchange graph can be mapped to a tree since there is exactly one path from P_i to P_j, with an order bounded by Eq. 1. An upper bound is analogous to a fully dense communication, such that $O(\log P)$ exchanges happen $O(\log P)$ times, which is equivalent to $O(\log^2 P)$. Table 1 shows `FMM` communication complexity for uniform domains.

Lemma 1. *A partition P' is added to the adjacency list of P iff for any dimension D $maxBound(P'_x) - maxBound(P_x) > \epsilon$ and $minBound(P_x) - minBound(P'_x) > \epsilon$*

Table 1. Communication complexity of FMM.

Reference	Communication complexity
Teng [27]	$\mathcal{O}\left(P(N/P)^{2/3}(\log N+\mu)^{1/3}\right)$
Lashuk *et al.* [26]	$\mathcal{O}\left(\sqrt{P}(N/P)^{2/3}\right)$
Lashuk *et al.* [16]	$\mathcal{O}\left(\log P+(N/P)\right)$
Yokota *et al.* [28]	$\mathcal{O}\left(\log P+(N/P)^{2/3}\right)$

Table 2. $\mathcal{HSDX}$ Algorithm communication symbols.

Symbol	Indication
P and P'	Local and global partitions
$\Omega(P)$	Subdomain boundary
$\zeta(\Omega(P))$	Direct neighbors of P
$\mathcal{T}$	Level-by-level communication adjacency graph

Algorithm 1. $\mathcal{HSDX}$ - Hierarchical Sparse Data Exchange

```
input  : A list $l_{in}$ of cells and destinations
output: A list $l_{out}$ of cells and sources
foreach $P'$ in $\Omega(P,\beta)$ do
    add(P',$\zeta(\Omega(P))$)
end
$\mathcal{T} \leftarrow$ BuildCommTree($\zeta(\Omega(P))$);
create distributed MPI graph topology;
foreach $l$ in $\mathcal{T}$.Levels do
    foreach $P'$ in $\zeta(\Omega(P))$ do
        reduce tree based on the bounding box and forward to P';
    end
    exchange meta data;
    call MPI_Neighbor_alltoallv;
end
```

4.3 Pairwise Exchange for Reducing Contention

It is observed at large scale that direct communication between sources and targets results in network contention which can be amortized by relaying multipoles through neighbor processes while utilizing the well-known pattern of N-Body interactions. Therefore, to mimic $O(\log P)$ complexity for boundary distributions, we implement a modified version of the well-known hypercube (butterfly) global communication scheme which starts out by the fine neighbor interactions depicted by $(P \oplus 2^i)$ and gets coarser as we move towards the $(\log P)$ step. This is clearly visualized in Fig. 4 using Extrae, a tool that uses different interposition mechanisms to inject probes into the target application so as to gather information regarding the application performance. During this work, the tool is used to better understand the performance of the application pertaining to the used communication techniques. In Fig. 4, the horizontal axis represents the visualized timeline and the vertical axis represents the MPI processes. The yellow colors declare communication links, whereas the blue colors represent computation and red color symbolizes MPI_Wait calls. One of the main advantages of carrying out communication in $(\log P)$ steps, as in is Fig. 4(a), is that subtrees received at intermediate stages can be asynchronously traversed, which otherwise cannot be done if communication is done with blocking collectives.

5 Performance Analysis

5.1 Experimental Setup

Our experiments are on Shaheen XC40, the rank 15 supercomputer according to the November 2016 Top500 list, located at King Abdullah University of Science and Technology. It has 196,608 physical cores and HPL performance of 5.537 PFlop/s. Each node is equipped with dual socket Intel Xeon E5-2698v3 16C 2.3 GHz and Cray Aries interconnect with dragonfly topology.

Throughout the following experiments, the underlying `FMM` code is compiled with the Laplace kernel, Cartesian coordinates, $P = 4$ (order of expansion) and spherical boundary distribution unless otherwise stated. Problems have been partitioned using the hybrid partitioning from Sect. 2.2. To demonstrate the effectiveness of the presented methodologies, we start by showing how optimal grain size for a specific problem is chosen, then assessing the scalability with the tuned granularity of communication. Then, results from using $\mathcal{HSDX}$ vs. existing communication reducing approaches are presented. Good scalability shows that an inordinate cost is not paid for intra-node communication, as opposed to the conventional bulk-synchronous approach, for which performance depends on the underlying network topology, the implementation of collectives like `alltoall` or `allgather`, the available memory size and bandwidth, and the frequency at which synchronization is triggered.

5.2 Communication Time for Different Granularities

In order to show the direct effect of asynchronous traversal on performance, we gradually vary the grain-size and measure the communication time, which is the most dominant factor at a large scale. Optimal granularity is a tuning parameter that varies with problem size, distribution and other factors as depicted by the average communication time in Fig. 6, where subtree size is gradually increased. The unit of communication is a subtree, which has 2, 4, 8, etc., cells as shown in the X-axis. The is a subset of the local essential `FMM` tree as we explain in Sect. 3. Communication time is measured by accumulating times of individual asynchronous sends and receives per process and taking their arithmetic mean.

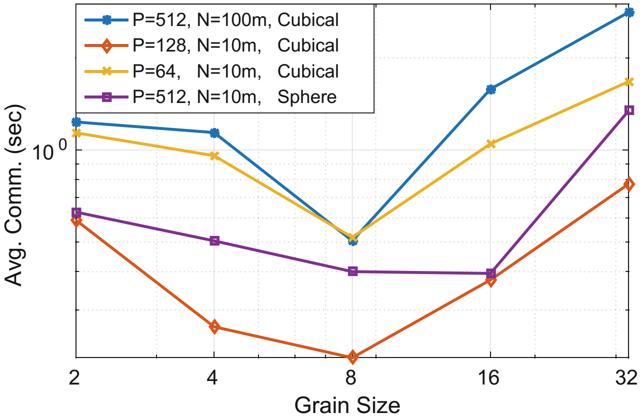


Fig. 6. Average communication time for different sizes and distributions as grain-size is varied.

The theoretical maximum subtree size is the entire LET. We stop at a certain threshold (32 in this case) because when it is increased further, a huge jump in time occurs. This is attributable to the change in communication protocol as per the Cray® MPICH specification from Eager Message to Rendezvous Message Protocol. When the message size exceeds a specific threshold (8 KB in this case), MPICH2 GNI NetMod alters the pathways towards a more relaxed algorithm for point-to-point inter-node messaging. A similar approach is developed in other MPI implementations like Open MPI and Intel® MPI. Hence, the remote tree traversal enables us to tune the performance by reducing the communication time enough to increase the impact of latency hiding.

5.3 Scalability of Spherical Boundary Distribution with $\mathcal{HSDX}$

In Fig. 7, we test the strong scalability at optimal grain size using $\mathcal{HSDX}$ for a large problem of 10^{10} particles. It follows that we have an efficient asynchronous communication when remote calls are non-blocking, have tunable granularity and when control is handed over to useful work rather than waiting immediately. To show this, we have integrated and compared several communication protocols within `ExaFMM` in Fig. 7. We note that $\mathcal{HSDX}$ is the closest to ideal scaling and has the advantage of fastest time-to-solution since it limits the inter-rack communication penalty on the dragonfly network by solely exchanging data through neighbors. By just looking at Fig. 7, it is hard to see that $\mathcal{HSDX}$ is at potential advantage for the exascale era. So we find it useful to present Table 3, which shows a more detailed analysis of the strong scalability. We notice a 6-fold increase in performance gain (from 3.87% to 23.44%) over the corresponding `MPI_Alltoallv` implementation as more cores are added. The parallel efficiency decreases, however, as the problem gets smaller while communication overhead prevails. Conventional $O(P)$ communication schemes stop

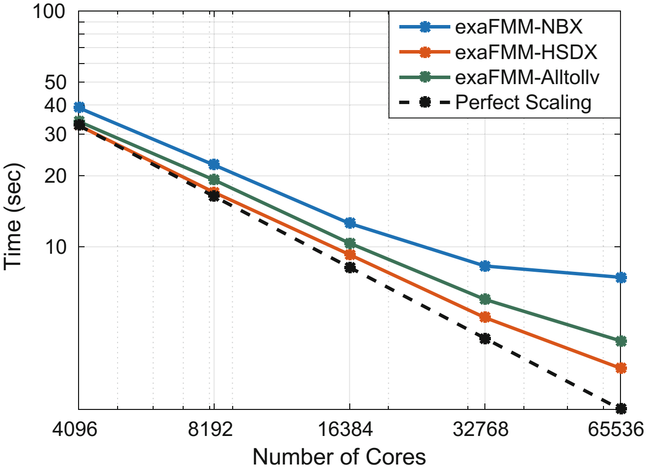


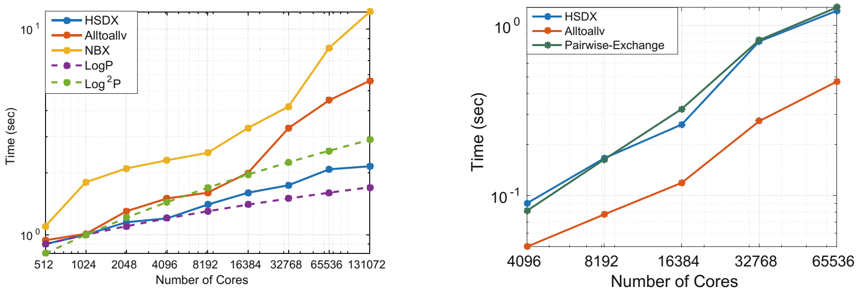
Fig. 7. Strong scaling across different communication protocols with 10^{10} particles.

Table 3. $\mathcal{HSDX}$ strong scalability analysis with `FMM`.

P	4, 096	8, 192	16, 384	32, 768	65, 536
HSDX	32.72	17.02	9.27	5.008	3.05
Rel. Speedup	1	1.92	3.53	6.53	10.70
Efficiency	-	0.96	0.88	0.81	0.66
Enhancement	3.87%	11.41%	10.55%	16.27%	23.44%

scaling after 2048 nodes (65,536 cores) of Shaheen XC40. According to our largest setup that has an input of $N = 10^{10}$, we have an update rate of approximately 10^9 particles/second.

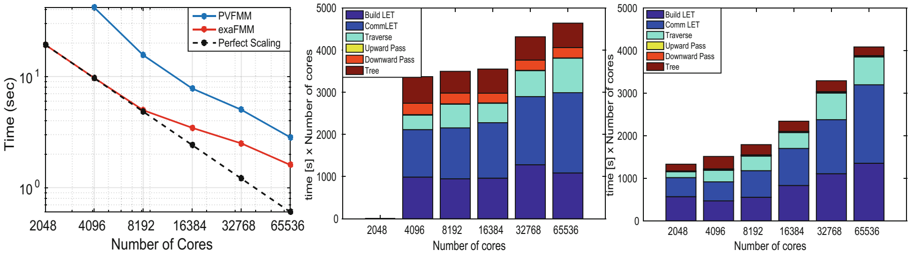
`PVFMM` is a large-scale `FMM` library that uses a kernel independent implementation, thus widens its target range of applications that require calculation of potential for elliptic kernels [29]. In this experiment, we attempt to compare the strong-scaling performance of `PVFMM` to our `ExaFMM` branch. It is worth noting that citing independent work is not meant to deem one superior to the other, but on the contrary, it is to give rise to our promising performance boosting strategies that tackle problematic communication and partitioning issues that are likely to arise in the near exascale era. In their most recent reports on `PVFMM`, Malhotra et al. [29] report perfect scalability up to 256 cores when running the Laplace kernel to compute potentials for 10^8 distributed on the surface of an ellipsoid. From that point onwards, communication cost starts to grow. They achieve 95% speedup corresponding too about 37% parallel efficiency. We switch to neighborhood collective communication presented in Sect. 4.2 for this comparison, since it vastly reduces network contention by propagating cells through direct neighbors only. Figure 8(a) shows consistent weak-scalability of communication



(a) Big example comparison of different communication protocols when weak-scaling 15m particles.

(b) Small example comparison of different communication protocols when weak-scaling 200k particles

Fig. 8. Communication scaling for big and small examples.



(a) `ExaFMM` vs `PVFMM` breakdown.

(b) `PVFMM` breakdown.

(c) `ExaFMM` breakdown.

Fig. 9. Strong scaling 2^{31} unknowns for sphere distribution and $P = 4$ and comparing `ExaFMM` while using $\mathcal{HSDX}$ communication and `PVFMM`.

over the conventional `MPI_Alltoallv` implementation. The presented approach shows a faster time-to-solution in `ExaFMM` vs. `PVFMM` when computing 2 billion unknowns as in Fig. 9. We cannot claim that scaling will persist indefinitely beyond the depicted number of cores, but when we have an exascale application that requires orders of magnitude larger problems that can fit in the machine's memory, we have a strong evidence of strong scalability.

5.4 Evaluation of Neighborhood Collective Communication Using $\mathcal{HSDX}$

Figure 8(a) compares $\mathcal{HSDX}$ using neighborhood collectives to $\mathcal{NBX}$ and `MPI_Alltoallv`. For the class of problems that constitute a hierarchically sparse data exchange defined in 1, $\mathcal{HSDX}$ is asymptotically bounded by the $c_1 \log P$ and $c_2 \log^2 P$. This behavior is shown for the boundary distribution solving Laplace Cartesian `FMM` kernels with $P = 4$ (order of expansion). However, the figure does not suggest that $\mathcal{HSDX}$ can generally replace its rivals; we still believe that $\mathcal{NBX}$ would outperform our algorithm in the general sparse data exchange, because it has the advantage of both $O(\log P)$ upper bound in addition to the use of a non-blocking barrier and synchronized sends [1].

Figure 8(b) weak scales a small example in order to reduce the effect of non-neighbor communication. The fact that $\mathcal{HSDX}$ and Pairwise exchange exhibit similar performance is anticipated since they almost have identical $logP$ behavior in such cases. They seem to lose herein against `MPI_Alltoallv` because of the initialization overhead included in communication time.

6 Conclusion

In this work, we propose algorithms that improve data locality, remote data access, and load-balance of the N-body problem. These algorithms contribute to producing an `FMM` solver that exploits communication redundancy and computation overlap. We show that Hilbert space-filling curves may not be the most optimal choice to partition boundary domain distributions. $\mathcal{HSDX}$ shows good strong and weak scalability for large adaptive hierarchically sparse problems, and falls within proven asymptotic time complexities. Shared memory parallelism is important to utilize resources within a node and to alleviate the problems with MPI resource management; thus we need to consider it in future implementations. We are working on improving $\mathcal{HSDX}$ so that it exploits the advantages of $\mathcal{NBX}$ to widen its range of use cases. As for application, we are intending to make the presented solver a part of an `FMM` preconditioner for the Poisson equation, which has variety of applications in diffusive and equilibrium processes in fluid dynamics and many other applications.

Acknowledgment. This work was supported by JSPS KAKENHI Grant-in-Aid for Young Scientists A Grant Number 16H05859. This work is partially supported by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" and "High Performance Computing Infrastructure" in Japan. The authors are grateful to the KAUST Supercomputing Laboratory for the use of the Shaheen XC40 system.

References

1. Hoefler, T., Siebert, C., Lumsdaine, A.: Scalable communication protocols for dynamic sparse data exchange. In: Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ser. PPoPP 2010, pp. 159–168. ACM, New York (2010)
2. Appel, A.W.: An efficient program for many-body simulation. SIAM J. Sci. Stat. Comput. **6**(1), 85–103 (1985)
3. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. J. Comput. Phys. **73**(2), 325–348 (1987)
4. Beatson, R., Greengard, L.: A short course on fast multipole methods. Wavelets Multilevel Methods Elliptic PDEs **1**, 1–37 (1997)
5. Lu, B., Cheng, X., Huang, J., McCammon, J.A.: Order N algorithm for computation of electrostatic interactions in biomolecular systems. Proc. Natl. Acad. Sci. **103**(51), 19314–19319 (2006)
6. Yokota, R., Bardhan, J.P., Knepley, M.G., Barba, L.A., Hamada, T.: Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns. Comput. Phys. Commun. **182**(6), 1272–1283 (2011)

7. Ohno, Y., Yokota, R., Koyama, H., Morimoto, G., Hasegawa, A., Masumoto, G., Okimoto, N., Hirano, Y., Ibeid, H., Narumi, T., et al.: Petascale molecular dynamics simulation using the fast multipole method on K computer. Comput. Phys. Commun. **185**(10), 2575–2585 (2014)
8. Rui, P., Chen, R.: An efficient sparse approximate inverse preconditioning for FMM implementation. Microw. Opt. Technol. Lett. **49**(7), 1746–1750 (2007)
9. Bédorf, J., Gaburov, E., Zwart, S.P.: A sparse octree gravitational N-body code that runs entirely on the GPU processor. J. Comput. Phys. **231**(7), 2825–2839 (2012)
10. Price, D., Monaghan, J.: An energy-conserving formalism for adaptive gravitational force softening in smoothed particle hydrodynamics and N-body codes. Mon. Not. R. Astron. Soc. **374**(4), 1347–1358 (2007)
11. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W., et al.: The landscape of parallel computing research: a view from Berkeley. Technical report UCB/EECS-2006-183, EECS Department, University of California, Berkeley (2006)
12. Warren, M.S., Salmon, J.K.: A fast tree code for many-body problems. Los Alamos Sci. **22**(10), 88–97 (1994)
13. Bédorf, J., Gaburov, E., Fujii, M.S., Nitadori, K., Ishiyama, T., Portegies Zwart, S.: 24.77 Pflops on a gravitational tree-code to simulate the Milky Way Galaxy with 18600 GPUs. In: Proceedings of the 2014 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2014)
14. Speck, R., Ruprecht, D., Krause, R., Emmett, M., Minion, M., Winkel, M., Gibbon, P.: A massively space-time parallel N-body solver. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 92. IEEE Computer Society Press (2012)
15. Winkel, M., Speck, R., Hubner, H., Arnold, L., Krause, R., Gibbon, P.: A massively parallel, multi-disciplinary barnes-hut tree code for extreme-scale N-body simulations. Comput. Phys. Commun. **183**(4), 880–889 (2012)
16. Lashuk, I., Chandramowlishwaran, A., Langston, H., Nguyen, T.-A., Sampath, R., Shringarpure, A., Vuduc, R., Ying, L., Zorin, D., Biros, G.: A massively parallel adaptive fast multipole method on heterogeneous architectures. Commun. ACM **55**(5), 101–109 (2012)
17. Zandifar, M., Abdul Jabbar, M., Majidi, A., Keyes, D., Amato, N.M., Rauchwerger, L.: Composing algorithmic skeletons to express high-performance scientific applications. In: Proceedings of the 29th ACM on International Conference on Supercomputing, ser. ICS 2015, pp. 415–424. ACM, New York (2015)
18. AbdulJabbar, M., Yokota, R., Keyes, D.: Asynchronous execution of the fast multipole method using charm++. arXiv preprint arXiv:1405.7487 (2014)
19. Salmon, J.K.: Parallel hierarchical N-body methods. Ph.D. dissertation, California Institute of Technology (1991)
20. Warren, M.S., Salmon, J.K.: A parallel hashed oct-tree N-body algorithm. In: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, pp. 12–21. ACM (1993)
21. Makino, J.: A fast parallel treecode with GRAPE. Publ. Astron. Soc. Jpn. **56**, 521–531 (2004)
22. Solomonik, E., Kalé, L.V.: Highly scalable parallel sorting. In: Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing (IPDPS), pp. 1–12 (2010)
23. Haverkort, H.: An inventory of three-dimensional Hilbert space-filling curves. arXiv preprint arXiv:1109.2323 (2011)

24. Dubinski, J.: A parallel tree code. New Astron. **1**, 133–147 (1996)
25. Warren, M.S., Salmon, J.K.: Astrophysical N-body simulations using hierarchical tree data structures. In: Proceedings of the 1992 ACM/IEEE Conference on Supercomputing, ser. Supercomputing 1992, pp. 570–576. IEEE Computer Society Press, Los Alamitos (1992)
26. Lashuk, I., Chandramowlishwaran, A., Langston, H., Nguyen, T.-A., Sampath, R., Shringarpure, A., Vuduc, R., Ying, L., Zorin, D., Biros, G.: A massively parallel adaptive fast multipole method on heterogeneous architectures. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (2009)
27. Teng, S.-H.: Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation. SIAM J. Sci. Comput. **19**(2), 635–656 (1998)
28. Yokota, R., Turkiyyah, G., Keyes, D.: Communication complexity of the fast multipole method and its algebraic variants. Supercomput. Front. Innov.: Int. J. **1**(1), 63–84 (2014)
29. Malhotra, D., Biros, G.: PVFMM: a parallel kernel independent fmm for particle and volume potentials. Commun. Comput. Phys. **18**(3), 808–830 (2015)