Accelerating Parallel Hierarchical Matrix-Vector Products via Data-Driven Sampling

Lucas Erlandson
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, Georgia, United States of America
lerlandson3@gatech.edu

Difeng Cai
Department of Mathematics
Emory University
Atlanta, Georgia, United States of America
dcai7@emory.edu

Yuanzhe Xi
Department of Mathematics
Emory University
Atlanta, Georgia, United States of America
yxi26@emory.edu

Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, Georgia, United States of America
echow@cc.gatech.edu

***Abstract*—Hierarchical matrices are scalable matrix representations particularly suited to the case where the matrix entries are defined by a smooth kernel function evaluated between pairs of points. In this paper, we present a new scheme to alleviate the computational bottlenecks present in many hierarchical matrix methods. For general kernel functions, a popular approach to construct hierarchical matrices is through interpolation, due to its efficiency compared to computationally expensive algebraic techniques. However, interpolation-based methods often lead to larger ranks, and do not scale well to higher dimensions. We propose a new *data-driven* method to resolve these issues. The new method is able to accomplish the rank reduction by using a surrogate for the global distribution of points. The surrogate is generated using a hierarchical data-driven sampling. As a result of the lower rank, the construction cost, memory requirements, and matrix-vector product costs decrease. Using state-of-the-art dimension independent sampling, the new method makes it possible to tackle problems in higher dimensions. We also discuss an *on-the-fly* variation of hierarchical matrix construction and matrix-vector products that is able to reduce memory usage by an order of magnitude. This is accomplished by postponing the generation of certain intermediate matrices until they are used, generating them just in time. We provide results demonstrating the effectiveness of our improvements, both individually and in conjunction with each other. For a problem involving 320,000 points in 3D, our data-driven approach reduces the memory usage from 58.75 GiB using state-of-the-art methods (762.9 GiB if stored dense) to 18.60 GiB. In combination with our on-the-fly approach, we are able to reduce the total memory usage to 543.74 MiB.**

I. Introduction

Many scientific and data applications are bottlenecked by a large scale matrix whose entries are defined by a kernel function evaluated at pairs of points from a given dataset. For many kernel functions, the resulting kernel matrices are dense. Therefore, with typical dense linear algebra methods, the storage cost associated with an n-by-n kernel matrix would be $O(n^2)$ and the cost of computing a matrix-vector product would be $O(n^2)$ as well. Hierarchical matrices provide a representation that can be used to provide asymptotically better storage and evaluation costs. In this paper, we discuss various bottlenecks associated with the construction and evaluation of hierarchical matrices, without requiring an analytical expansion of the kernel function.

A. Motivation

It is well known that hierarchical matrix methods can provide asymptotic speedup when compared with dense linear algebra methods. However, the cost of deriving hierarchical representations can be significant, especially when the approximation rank is much larger than the actual rank. This paper will focus on $\mathcal{H}^2$ matrices, which can be constructed, stored, and applied in optimal $O(n)$ time and space enabled by the nested basis property [1], [2], [3]. A simpler hierarchical structure, associated with $\mathcal{H}$ matrices [4], [5], [2], [3], does not require nested bases and has a suboptimal cost of $O(n \log n)$ in storage and matrix-vector products. Among these two types of hierarchical matrices, it is particularly difficult to create a black-box high performance implementation of $\mathcal{H}^2$ matrices.

In practice, the construction of a hierarchical matrix is much more expensive than multiplying it by a vector. As an example, the construction cost using algebraic techniques is at least quadratic, while the cost of computing a matrix-vector product associated with the resulting hierarchical format is near linear. Interpolation-based methods provide a general, yet efficient, way to construct $\mathcal{H}^2$ matrices. They can bring the construction costs down to $O(n)$ and are used to solve a wide range of problems [6], [7]. However, one issue with interpolation-based methods is that their costs have very large prefactors. This is because the low-rank factors in the resulting hierarchical matrices have much larger rank than needed for a given approximation accuracy. The other issue with interpolation-based methods is that their costs scale exponentially with respect to the number of spatial dimensions. Thus, these methods rapidly lose their efficiency in higher dimensions. The primary motivation of the *data-driven* method proposed

in this paper is to achieve more efficient scaling while being able to handle equally as general problems as interpolation-based methods.

A commonality among hierarchical matrix implementations is that all the low-rank factors are calculated and stored during the construction, and are later used in performing matrix-vector products. This is in contrast to the fast multipole method (FMM) where the hierarchical low-rank format is generated just in time for use, and discarded after use [8]. This, on the other hand, makes FMM less efficient when a large number of matrix-vector multiplications need to be performed, for example, in the iterative solution of linear systems. In this case, the hierarchical representation has to be computed from scratch in each iteration. In order to trade off the memory and computation involved, we take advantage of the special structure of low-rank factors produced by the SMASH algorithm [9] and propose an *on-the-fly* approach when using the hierarchical format. Since most factors produced by this algorithm are submatrices of the kernel matrix, instead of storing these factors explicitly, we only store the corresponding row and column indices. This significantly reduces the memory cost and these submatrices can be rapidly assembled in parallel whenever needed.

We propose new methods to alleviate the bottlenecks that arise in $\mathcal{H}^2$ matrices and hierarchical matrices in general. In summary:

- We introduce a new data-driven sampling method, which produces lower ranks for $\mathcal{H}^2$ matrices and achieves a speedup up to 10^4 in high dimensions when compared to the interpolation-based method;
- We discuss an on-the-fly handling of the matrix-vector products, which reduces memory consumption by an order of magnitude;
- We provide parallel numerical experiments, demonstrating the effectiveness of the above contributions.

B. Background

1) Hierarchical Matrices: There exists much literature discussing hierarchical matrices and their applications. Surveys can be found in [3], [2], [10], [11]. A variety of packages are described in [12], [13], [14]. The ideas behind hierarchical matrices can be traced back to [4], [15], [8], [5], where researchers sped up the evaluation of n-body gravitational potentials [4] or Coulomb potentials [8], and the iterative solution of boundary integral equations [15], [5]. Mathematically, the computational task boils down to computing the matrix-vector product involving a dense matrix associated with certain kernel function $\mathcal{K}(x, y)$ evaluated at a set of points $X = \{x_1, \ldots, x_n\}$:

$$A = [\mathcal{K}(x_i, x_j)]_{i,j=1:n}.$$

For large problems, a straightforward calculation suffers from a prohibitive $O(n^2)$ complexity in time and space. The above mentioned methods circumvent the computational bottleneck by compressing certain blocks in the original matrix and bring the cost to be near linear. The same principle has since been generalized into the algebraic framework of hierarchical matrices, in particular $\mathcal{H}$ and $\mathcal{H}^2$ matrices. The algebraic counterparts can handle a larger class of kernel functions [2], [3] and approximate A explicitly with a hierarchically low-rank matrix $\hat{A}$. For a prescribed accuracy tolerance, storing and multiplying a hierarchical matrix by a vector has near linear scaling with the matrix dimension.

The construction of a hierarchical matrix representation for a given kernel matrix involves several steps. For a general dataset X where points may not be uniformly distributed, an adaptive partitioning of the dataset is first performed, to build up the hierarchy and identify low-rank blocks for the associated kernel matrix. That is, the dataset is divided geometrically and recursively until the number of points in each resulting subset is small enough (such that performance is optimized). Meanwhile, a tree is generated to encode the hierarchical structure of the partitioning, where each node in the tree corresponds to a subset of points in the partitioning. For example, the root node corresponds to the entire set of points, and its children correspond to subsets of points after the initial partitioning. We use X_i throughout the paper to denote the set of points associated with node i. Two nodes i and j are called *well-separated* if the corresponding point sets X_i and X_j are well-separated by a certain criterion (cf. [2], [9]). The included experiments consider i and j to be well-separated if the maximum diameter of X_i and X_j is less than 0.7 times the distance between the midpoints of X_i and X_j. Any submatrix associated with well-separated clusters is assumed to be numerically low-rank and can be well-approximated by a low-rank matrix. Such a submatrix is often referred as a *farfield block*. A hierarchical matrix approximation $\hat{A}$ replaces the farfield blocks in the original matrix A by low-rank approximations, i.e.,

$$A_{i,j} \approx \hat{A}_{i,j} = U_i B_{i,j} V_j^T \tag{1}$$

for well-separated nodes i and j, where $A_{i,j}$ and $\hat{A}_{i,j}$ denote the submatrices of A and $\hat{A}$ respectively, associated with subsets X_i and X_j. The matrices $B_{i,j}$ connecting two basis matrices U_i and V_j are called *coupling matrices*. The above structure gives rise to $\mathcal{H}$ matrices, which have $O(n \log n)$ complexity in storage and matrix-vector products. To further reduce the complexity to $O(n)$, a more complicated structure is needed, one using the nested basis property. That is, if node p is the parent of nodes i, j, k, then the corresponding basis matrices U, V are nested in the following way:

$$U_p = \begin{bmatrix} U_i R_i \\ U_j R_j \\ U_k R_k \end{bmatrix}, \quad V_p = \begin{bmatrix} V_i W_i \\ V_j W_j \\ V_k W_k \end{bmatrix},$$

where R and W matrices are called *transfer matrices* and are of size $O(1)$. This nested basis property enables one to only store transfer matrices instead of all basis matrices explicitly, as a parent node can be constructed from its children. Hierarchical matrices with such a nested basis property are called $\mathcal{H}^2$ *matrices*. Meanwhile, when two sets of points are close to each other, the corresponding matrix block is called

a *nearfield block* and is not approximated. The hierarchical partitioning of the entire set of points ensures that the nearfield blocks are only associated with leaf nodes and the submatrix that consists of all the nearfield blocks is sparse (cf. [2], [3], [16]). Therefore, a reduction in cost from $O(n^2)$ to $O(n \log n)$ or $O(n)$ is achieved by storing only the nearfield blocks and the low-rank approximations for farfield blocks. All the basis matrices, transfer matrices and nearfield blocks are called the *generators* of $\mathcal{H}^2$ matrices.

To construct $\mathcal{H}^2$ representations, one needs a way to approximate farfield blocks and simultaneously maintain the nested basis property. For FMM and its variants [15], [8], [16], expansions such as Taylor expansions or spherical harmonic expansions are used due to high accuracy and low computational complexity. The so-called kernel independent fast multipole method [17], [18] derives factorizations by solving ill-posed integral equations. One limitation of these methods is that they are only valid for special kernel functions, i.e., the fundamental solutions of certain constant coefficient partial differential equations, such as the Laplace equation, low-frequency Helmholtz equations, the Stokes equation, etc. To handle general kernel functions, a common technique that allows for black-box kernel independent implementations is polynomial interpolation. Due to the efficiency and generality of interpolation, interpolation-based hierarchical matrix methods have been used for solving many types of problems [2], [6], [19], [20], [3], [9].

2) Interpolation-Based Construction: Interpolation was first introduced for $\mathcal{H}^2$ matrices in [2], [6] as a replacement for Taylor expansions, as Taylor expansions require evaluation of the derivatives of the desired functions which may have numerical overflow or underflow issues (cf. [16]). Conversely, interpolation only requires evaluations of the kernel function, making it ideal for constructing hierarchical matrices for arbitrary user-defined kernel functions. Compared to algebraic techniques, interpolation is able to provide explicit formulas for all low-rank factors in the hierarchical representations and hence the total computational cost is small. We review the basic idea of interpolation-based construction below.

The use of polynomial interpolation (cf. [9]) yields the following separable approximation for $\mathcal{K}(x, y)$

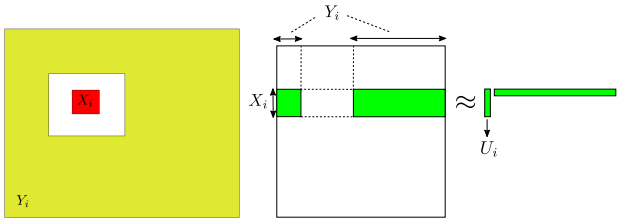
$$\mathcal{K}(x, y) \approx \sum_{k=1}^{r} p_k(x) \mathcal{K}(x_k, y),$$

where x_k are interpolation points and p_k are the associated Lagrange polynomials ($k = 1, \ldots, r$). The separable approximation above automatically induces a low-rank approximation of the entire farfield block for node i:

$$\begin{aligned} A_i &:= [\mathcal{K}(x, y)]_{\substack{x \in X_i \\ y \in Y_i}} \\ &\approx \left[p_1^{(i)}(x), \cdots, p_r^{(i)}(x)\right]_{x \in X_i} \left[\mathcal{K}(x_k^{(i)}, y)\right]_{\substack{k=1:r \\ y \in Y_i}}, \end{aligned} \tag{2}$$

where Y_i denotes the set of *all* points that are well-separated from X_i. Thus, the column basis U_i can be chosen as

$$U_i = \left[p_1^{(i)}(x), p_2^{(i)}(x), \cdots, p_r^{(i)}(x)\right]_{x \in X_i}. \tag{3}$$



(a) X_i (red) and its farfield Y_i (yellow).

(b) Corresponding farfield block A_i (green) and the low-rank approximation.

Fig. 1. Demonstration of the farfield for node i.

See Fig. 1 for a pictorial demonstration.

Despite its generality and computational efficiency, interpolation usually does not yield the optimal rank in the approximation. That is, the approximation rank r in (2) can be much larger than the optimal rank under a prescribed tolerance. This is due to the fact that interpolation does not fully exploit the information from the kernel matrix, as one can see from (3) that the basis U_i is independent of the kernel function $\mathcal{K}$.

A more serious limitation of interpolation is that it suffers from the curse of dimensionality. The cost of interpolation-based construction methods scales exponentially with the number of dimensions, making them a poor choice for problems involving more than a few dimensions. For example, in d dimensions, interpolation over a tensor grid with p points per direction yields p^d interpolation points in total, i.e., an approximation rank $r = p^d$ in (2). Hence, we see that interpolation-based hierarchical low-rank approximations quickly lose their efficiency in high dimensions.

II. Methods

In this paper, we propose two novel methods for use in hierarchical matrix packages:

1) a new data-driven construction of hierarchical matrices with nested bases;
2) a memory efficient on-the-fly approach for matrix-vector products.

The data-driven approach breaks the curse of dimensionality seen by interpolation-based methods. Our experiments show that the data-driven approach yields blocks of lower rank (hence lower storage) for the same approximation error. Such a comparison can be seen in Fig. 2, where it is visible that the rank achieved by the data-driven method for the farfield nodes is significantly lower than the rank achieved by the interpolation based method. The on-the-fly approach further reduces the memory usage of hierarchical matrix representations by taking advantage of the special structure in coupling matrices [9]. By postponing the generation of certain matrices until they are used, the on-the-fly approach reduces memory usage, allowing larger problems to be solved.

A. Data-Driven Hierarchical Construction

1) Overall Idea: The data-driven $\mathcal{H}^2$ matrix approach employs a submatrix of the kernel matrix as the basis matrix for a farfield block. For example, for a farfield block A_i as

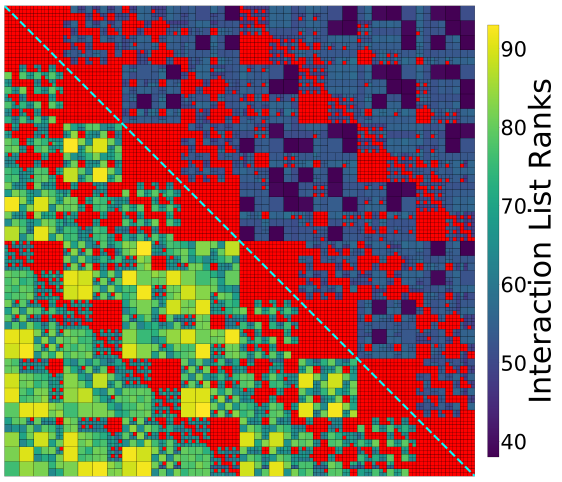


Fig. 2. A comparison of the rank of the bases produced by the interpolation-based (lower triangular part) method and the data-driven (upper triangular part) based method for 10,000 points randomly distributed in a cube for 1e-7 relative error for the Coulomb kernel. Red denotes nearfield interactions.

shown in Fig. 1, the column basis in the data-driven case is $U_i = \mathcal{K}(X_i, Y_i^*)$, where Y_i^* is a small subset ($O(1)$ in size) of Y_i. Since each Y_i contains $O(n)$ points and there are $O(n)$ nodes in total, naive sampling for each Y_i leads to at least $O(n^2)$ cost for deriving all basis matrices. Therefore, it is mandatory to sample Y_i^* hierarchically so as to lower the total cost to $O(n)$. Since the data-driven approach takes into account the kernel matrix, it enjoys an improved efficiency compared to interpolation-based methods. Particularly, the advantages of the data-driven approach are more prominent for high dimensional problems.

2) Nyström Sampling: The Nyström method [21] is a popular approach for deriving low-rank approximations via sampling and has been widely used in machine learning. Given point sets S, T, let $K_{S,T}$ denote a matrix with entries $\mathcal{K}(s,t)$ for $s \in S$, $t \in T$. To approximate the kernel matrix $K_{X,X}$ by a low-rank factorization, the Nyström method computes a set S that is much smaller compared to the size of X, and constructs the following approximation

$$K_{X,X} \approx K_{X,S} K_{S,S}^{+} K_{S,X},$$

where $K_{S,S}^{+}$ denotes the pseudoinverse of $K_{S,S}$. Once S is selected, $K_{X,S}$ serves as a column basis for the low-rank approximation. The original Nyström method chose S to be a subset of X associated with randomly chosen indices. The choice of S significantly affects the approximation accuracy and computational efficiency of Nyström methods. Various sampling strategies have been proposed to improve the performance of the original Nyström method [21], such as leverage score based sampling [22], [23], k-means based sampling [24], anchor net based sampling [25], etc. In this paper, we adopt the anchor net based sampling in [25] due to its efficiency for high dimensional problems.

3) Bottom-to-Top Sweep: The key to avoiding a quadratic sampling complexity is to sample the entire dataset hierarchically. This hierarchical sampling procedure starts with a bottom-to-top sweep following the partition tree. The anchor net Nyström method [25] is used to select the sample points inside each subset. We first sample over the points associated with each leaf node and then pass the samples to the parent. Since there are $O(1)$ points in each node at the leaf level, the cost associated with each leaf node is $O(1)$. Note that a parent node has $O(1)$ children and each child passes $O(1)$ samples, so the parent of each leaf node is associated with a new set of points with $O(1)$ size. Next we perform the same operation for each parent node as in the leaf level. That is, we perform sampling over the new set of points for each parent node and pass the output to the next level. The operation is repeated until we reach the root node. Since the cost associated with each node is $O(1)$, the total cost for the bottom-to-top sweep is $O(n)$. An illustration of the samples selected at the leaf level for a 2D dataset is shown in Fig. 3a.

4) Top-to-Bottom Sweep: The top-to-bottom sweep is then performed on the samples from the farfield associated with each node. We perform sampling over each such subset and pass the output to the children nodes along the partition tree. Since computing samples at each node has $O(1)$ complexity, the total cost for this sweep is also $O(n)$. An illustration of the samples from the farfield of a block for a 2D dataset is shown in Fig. 3b.

Note that the sampling step is only performed on points in the original set, and is independent of the kernel function and the kernel matrix. While sampling has previously been used in hierarchical methods, to the best of our knowledge, this is the first time that sampling techniques have been used in a hierarchical way. An outline of the hierarchical sampling is shown in Algorithm 1.

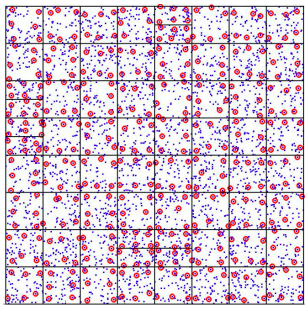
To summarize, the proposed data-driven method enjoys the following features:

1) allows black-box kernel independent construction of the hierarchical low-rank format;
2) provides optimal $O(n)$ complexity for the construction of nested bases, where n is the number of given points;
3) is valid for high dimensional problems (more than 3 dimensions);
4) achieves lower rank than interpolation-based methods for the same accuracy.

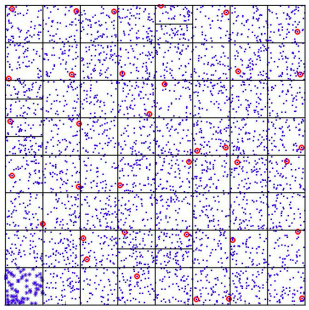
B. On-The-Fly Matrix-Vector Products

State-of-the-art methods for performing $\mathcal{H}^2$ matrix-vector products calculate the coupling matrices $B_{i,j}$ during the construction of the matrix. The $B_{i,j}$ matrices are only used to perform matrix-vector products. In the new $\mathcal{H}^2$ on-the-fly memory mode, rather than calculating the $B_{i,j}$ matrices during the construction of the matrix, they are calculated as needed in lines 9 and 15 of Algorithm 2.

Existing hierarchical matrix implementations calculate and store all the generators during the construction of the matrix, which will then be (re)used later. While the memory consumption scales linearly, we observe that the majority of the memory consumption arises from the storage of the coupling



(a) Samples X_i^* (red circles) from all leaf nodes i.



(b) Samples Y_i^* (red circles) from the farfield set Y_i for X_i (blue stars) in the bottom left corner.

Fig. 3. Illustration of the hierarchical sampling.

Algorithm 1 Data-driven hierarchical sampling

procedure HIERARCHICAL_SAMPLE(X)
Output: Y_i^*
  for all i **do**
    Set Y_i^* to be empty
    if i is a leaf node **then**
      Set $X_i^* = X_i$, (points associated with node i)
    else
      Set X_i^* to be empty
    end if
  end for
  for each node i from bottom to top **do**
    Set $X_i^* = \text{Sampling}(X_i^*)$
    Add X_i^* to X_p^*, the set associated with parent p
  end for
  for each node i from top to bottom **do**
    Set $Y_i^* = \bigcup X_j^*,\ j \in$ interaction list of i
    Update $Y_i^* = \text{Sampling}(Y_i^*)$
    Add Y_i^* to Y_c^* for each child c of i
  end for
end procedure

matrices $B_{i,j}$. Since $B_{i,j}$ is a submatrix of the original kernel matrix, memory consumption can be significantly reduced by storing the indices instead of the whole matrix $B_{i,j}$. The use of the on-the-fly memory scheme enables problems an order of magnitude larger to be tackled compared to traditional approaches.

III. Implementation Details

In this section, we describe our shared memory parallel implementation for comparing the performance resulting from data-driven sampling vs. interpolation and on-the-fly mode vs. normal memory mode. Our description is in two major parts: the construction of the $\mathcal{H}^2$ matrix and the application of the matrix via matrix-vector products. The coarsest level of parallelism arises directly from the structure of the partition tree. During the bottom-to-top sweeps of the tree, only information from the descendants of a node is required to calculate the generators associated with that node. Thus, all nodes on the same level of the tree can be processed in parallel. Similar parallelism is found in the top-to-bottom sweeps where all nodes on a given level can be processed in parallel. Finally, certain operations require a "horizontal sweep," where there is no dependency on the ordering of the computation, and thus all nodes can be processed simultaneously.

Algorithm 2 $\mathcal{H}^2$ matrix-vector product

procedure $\mathcal{H}^2$MAT-VEC($b, U, V, B, W, R,$ tree)
Output: $y = \hat{A}b$
  for each leaf node i **do**
    $q_i = V_i^T b_i$
  end for
  for each non-leaf node i from bottom to top **do**
    $q_i = \sum_{c \in \text{children of } i} W_c^T q_c$
  end for
  for each non-leaf node i **do**
    $g_i = \sum_j B_{i,j} q_j,\ \forall j \in$ interaction list of i
  end for
  for each non-leaf node i from top to bottom **do**
    $g_c = g_c + R_c g_i$
  end for
  for each leaf node i **do**
    $y_i = U_i g_i + \sum_j B_{i,j} b_j,\ \forall j \in$ nearfield of i
  end for
end procedure

A. $\mathcal{H}^2$ Matrix Construction

Our construction phase has two parts. First, the construction of the tree and second, the construction of the matrix. The tree construction is conducted in a divide-and-conquer manner, where initially the entire set of points is considered. This set is then partitioned, where each partition can be considered independently and in parallel with others. If a given node contains more than a heuristically determined number of points, this process is recursed. During the tree construction, the parent of each node is tracked, and after the construction this information is used to determine the children associated with each node, as well as other hierarchical information such as which level each node is on. Finally, once the construction of the hierarchy information is completed, the determination of which nodes are well-separated is performed.

The determination of well-separated nodes is completed via a recursive method, which starts by considering the interaction of the root node with itself. If both nodes being considered are well-separated, they are added to each other's *interaction list*. A node's interaction list corresponds to the nodes that are in the farfield of the node, but not in the farfield of the node's parent. Otherwise, if both are leaf nodes, they are added to each other's nearfield list. If one or both have children, the process is repeated among the children.

Once the hierarchy information has been calculated, we can perform the sampling given in Algorithm 1, which is independent of the kernel. Algorithm 1 consists of a bottom-to-top sweep and a top-to-bottom sweep. These sweeps can be

performed using the parallelization method described above, by considering all of the nodes on a level in parallel.

The construction of the basis matrices and the indices associated with coupling matrices is completed in a bottom-to-top sweep, and can be performed in parallel for all nodes at a given level. If the on-the-fly memory mode is not being utilized, the calculation of the coupling matrices is performed. This can be performed completely in parallel, by calculating the interaction between every node with the nodes in its interaction list. Note that our implementation uses a separate data structure to store the $B_{i,j}$ matrices. This is due to the fact that if the interactions between nodes are considered as a matrix, the matrix would be very sparse. Thus, our data structure consists of a sparse matrix of integers, and a sequence of dense matrices. The sparsity of the sparse matrix corresponds to the interactions between nodes, with the value of the element at (i, j) providing the linear index into a vector of dense matrices for $B_{i,j}$. Notably, this data structure is a C++ class with a matrix-free interface, and thus can be used for on-the-fly mode as well. For on-the-fly mode, rather than populating all the $B_{i,j}$ matrices, they are calculated as needed. In the symmetric case, only half of the $B_{i,j}$ matrices are required, as $B_{i,j} = (B_{j,i})^T$.

B. Matrix-Vector Product

The matrix-vector product consists of five stages, as seen in Algorithm 2. First, a horizontal sweep at the leaf node is performed, during which all leaf nodes can be considered in parallel. Then, a bottom-to-top sweep is done, which can take advantage of the bottom-to-top parallelization scheme mentioned at the beginning of Section III. A horizontal sweep is then performed, applying the coupling matrices associated with each node to the vector. Every application of $B_{i,j}$ can be considered in parallel. In the on-the-fly case, the matrix-vector product call to the class described above will calculate and apply $B_{i,j}$ at this point, however in the other memory modes $B_{i,j}$ is retrieved from the data structure and applied. After the horizontal sweep, a top-to-bottom sweep is performed, propagating the farfield-interactions (calculated via the interaction list) to the children. Finally, a horizontal sweep over the leaf nodes is performed, taking into account the nearfield/direct interactions.

C. Kernel Evaluation

Many of the calculations performed during the construction and application of hierarchical matrices are kernel evaluations. Thus, it is paramount to have efficient kernel evaluations. These evaluations can be accelerated by exploiting the SIMD instructions present in modern CPUs. Note that, like for direct interactions, the calculation of $B_{i,j}$ involves two clusters of points and there is an upper limit on the number of pairs of points for which the kernel evaluation will be performed. The maximum number of points per node tends to be on the order of hundreds.

D. Data-Driven Sampling

As shown in Algorithm 1, data-driven sampling is performed via a bottom-to-top sweep and then a top-to-bottom sweep. In these sweeps, nodes at the same level of the tree can be processed in parallel. Note that during the sampling step, where Nyström sampling is performed by finding the points nearest to a set of lattice points, Euclidean distances between the lattice points and the considered points are calculated.

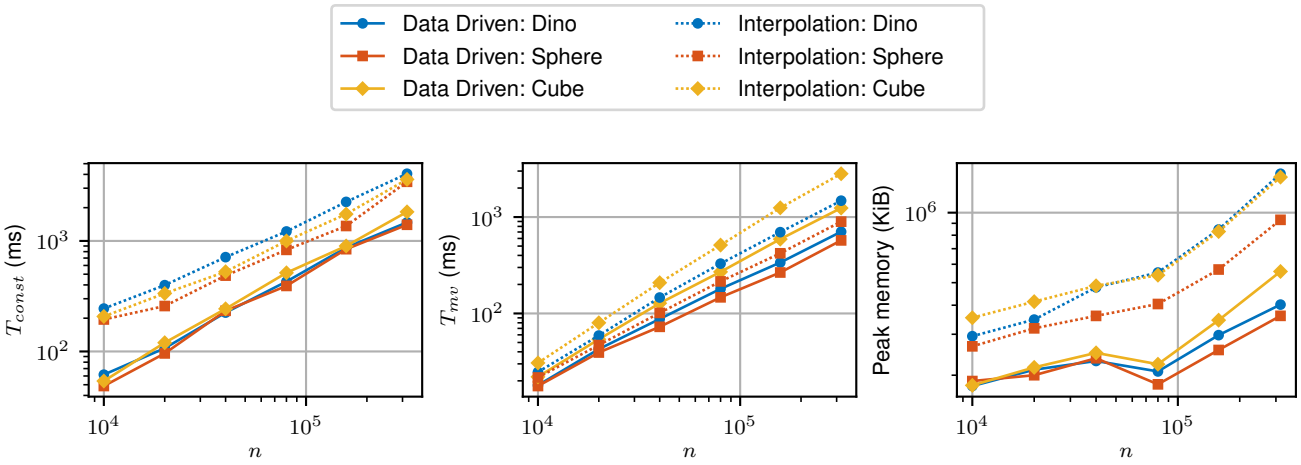
IV. Experimental Setup

We report experimental timings for the $\mathcal{H}^2$ matrix construction and matrix-vector products. The test sets of points used of these experiments are randomly generated over the surface of a sphere (`sphere`), in the volume of a cube (`cube`), and over the surface of a dinosaur (`dino`). The dinosaur test set is a complex 3D pointcloud, which is used to demonstrate the ability for these methods to handle highly non-uniform data [9], [26]. The timings of the algorithms were measured in separate parts, T_{const}, the $\mathcal{H}^2$ matrix construction time, and T_{mv}, the time required to perform a single matrix-vector product, both in milliseconds. The construction cost only occurs once, and can be amortized over many matrix-vector products. The experiments were conducted on a single node with 128 GB of memory and two Intel Xeon E5-2680 v4 CPUs, which have a base clock speed of 2.4 GHz and 14 cores. Unless otherwise noted, experiments were performed with 14 OpenMP threads and using the Coulomb kernel $1/||x - y||_2$. The relative error is measured as $||z - \hat{z}||_2/||z||_2$, where $\hat{z}$ is composed of 12 rows sampled randomly from the $\mathcal{H}^2$ matrix-vector product, and z contains the corresponding rows in the exact matrix-vector product.

V. Numerical Results

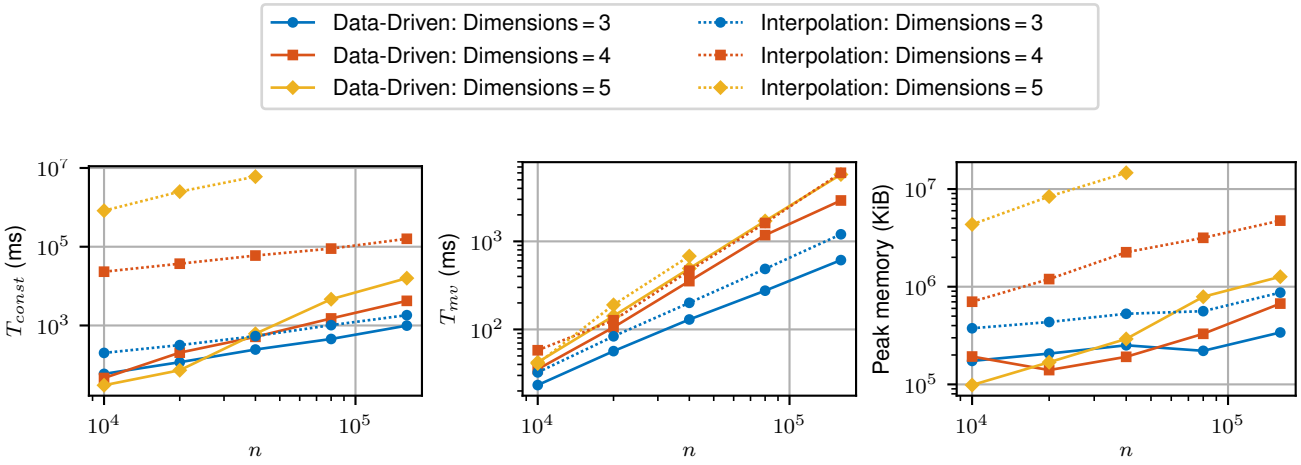
Fig. 4a shows that the point distribution does not have a notable impact on the construction time using on-the-fly memory mode. Fig. 4b shows that the asymptotic scaling remains roughly the same for the different distributions. In Fig. 4c, we see that the Sphere distribution requires less memory than the Cube distribution. This is due to the relative sparsity of the Sphere distribution, as the points are not uniformly distributed in the 3D domain, and there exists much empty space and fewer nearfield nodes, reducing the number of dense matrices required to be stored. The inflection point in memory usage is a result from the generally effective, but not optimally tuned, parameters of the construction method. Fig. 4b and Fig. 4c show that the data-driven method's matrix-vector products scale the same as, or better than, interpolation, and have a lower prefactor, while using less memory.

Fig. 5 demonstrates the scaling of the data-driven method with respect to the number of dimensions when using the on-the-fly memory mode. It is clear from Fig. 5a and Fig. 5c that the construction and memory usage scale significantly better in the data-driven case compared to interpolation-based methods. For example, with 160,000 points, going from three to four dimensions gives a 87.05 fold increase in construction time and 5.46 fold increase in peak memory usage for the interpolation-based methods, while the data-driven method increased only 4.25, and 1.87 times, respectively. Note that due to time and memory constraints, the interpolation-based



(a) Construction time (ms) (b) Matrix-vector product time (ms) (c) Peak memory usage

Fig. 4. Data-driven and interpolation-based methods on a variety of distributions uising on-the-fly memory mode for the Coulomb kernel. The relative accuracy for all tests is around 1e-8.



(a) Construction time (ms) (b) Matrix-vector product time (ms) (c) Peak memory usage

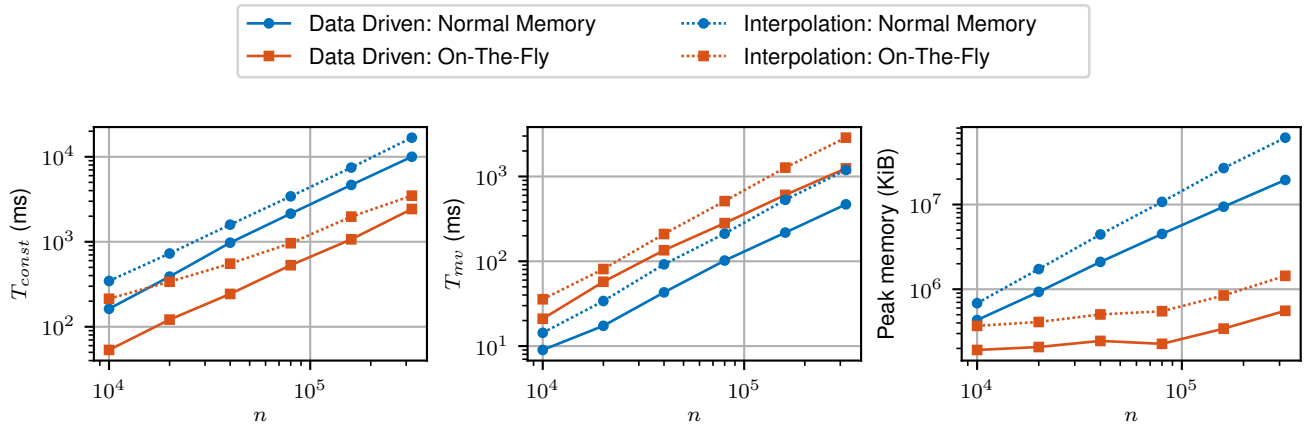
Fig. 5. Data-driven and interpolation-based methods on points in increasing dimensions using the on-the-fly memory mode for the Coulomb kernel with points in the volume of a hypercube, where the relative accuracy is fixed around 1e-8.

method was not tested for problems involving more than 40,000 points in five dimensions.

Fig. 6 details the cumulative effect of the new basis calculation via the data-driven method and the on-the-fly memory mode. We observe that the effects are cumulative, where using the data-driven method and on-the-fly memory at the same time results in the lowest memory usage and construction time. The memory scaling using on-the-fly memory is slightly better than that in the normal memory mode, as the normal memory mode scales with both the size and the number of farfield blocks while the on-the-fly memory mode scales only with the size of the blocks. As can be seen from Table I, the total memory reduction is from 58.75 GiB to 543.74 MiB, for the case of 320,000 points.

Fig. 7 displays the scaling of on-the-fly methods with the number of OpenMP threads for 1,000,000 points. Normal memory mode was not tested, as interpolation in normal memory mode requires more memory for this problem size than what is available. While the scaling of the construction seen in Fig. 7a is sub-linear, due to the difficulty of parallelizing the upper levels of the recursive bisection, it can be seen in Fig. 7b that the matrix-vector products have near linear scaling in both cases. Fig. 7c demonstrates that the memory usage increases slightly with the number of threads, p. Each thread stores only one $B_{i,j}$ matrix at a time; thus, the concurrent memory usage is $p \cdot \text{size}(B_{i,j})$.

Fig. 8 shows a comparison of the data-driven and interpolation-based methods as a function of the approximation error. This demonstrates that the data-driven method with the on-the-fly memory mode, for a given relative error, requires lower construction time, memory usage, and matrix-vector time. This holds true even in the low accuracy case, where interpolation is known to be the standard choice. These results demonstrate the effectiveness of the data-driven method across



(a) Construction time (ms) (b) Matrix-vector product time (ms) (c) Peak memory usage

Fig. 6. The data-driven and on-the-fly methods tested on increasing number of points for the Coulomb kernel with points in the `cube` distribution, where the relative accuracy for all tests is around 1e-8.

TABLE I
TIMINGS AND MEMORY CONSUMPTION USING DATA DRIVEN AND INTERPOLATION-BASED METHODS.

n	Basis	Memory	T_{const} (ms)	T_{mv} (ms)	Memory (KiB)
320,000	Interpolation	Normal	16789	1193	61603893
320,000	Interpolation	On-The-Fly	3488	2869	1440420
320,000	Data Driven	Normal	10011	469	19507675
320,000	Data Driven	On-The-Fly	2430	1245	556789

a wide range of accuracy, in addition to the number of points. The performance gap becomes even larger as the accuracy increases.

Fig. 9 shows the generality of the new data-driven method by demonstrating the method for different kernel functions using the on-the-fly memory mode. The cubed Coulomb kernel is given by $1/||x-y||_2^3$, the exponential kernel by $exp(-||x-y||_2)$, and the Gaussian by $exp(-||x-y||_2^2/0.1)$. It can be seen that, in most cases, the plots for the different kernels are nearly indistinguishable, demonstrating the generality of the new method. With the exception of the Gaussian kernel, the scaling for the different kernels are all nearly identical.

VI. Discussion

A. Data-Driven Basis Construction

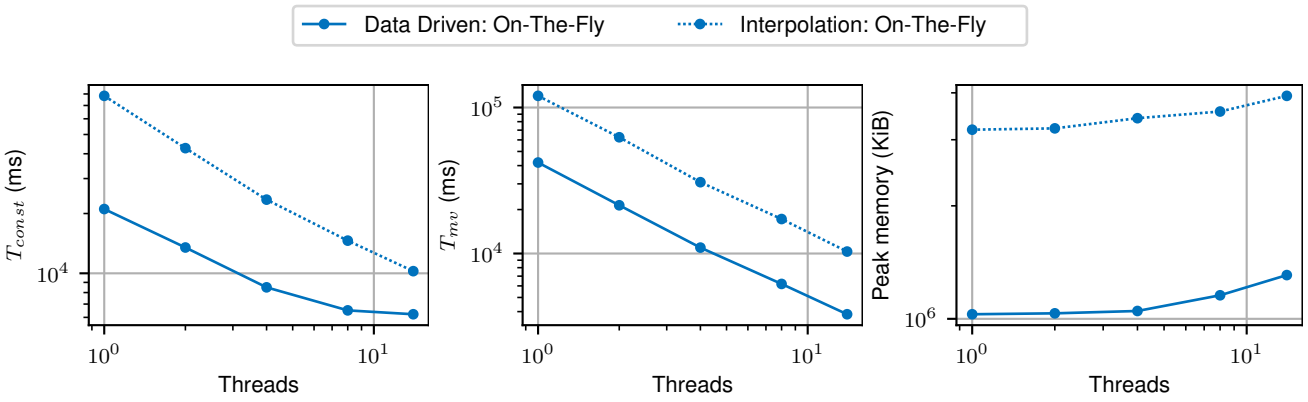
From Section V, the benefits of the data-driven method are numerous. Compared to the interpolation-based method, the data-driven method uses much less memory, as well as reduces the time taken by the matrix-vector product and $\mathcal{H}^2$ matrix construction. The majority of the time associated with the construction of the hierarchical matrix using the data-driven method comes not from the calculation of the basis, but rather the sampling. During the matrix-vector products, the majority of the time spent is in calculating the direct or nearfield interactions. Fortunately, the hierarchical sampling is done independently of the kernel, and depends only on the points; thus, for applications where multiple kernels must be used on the same data, the cost of sampling is amortized. As seen in Fig. 4 and Fig. 9, the data-driven method is equally general as interpolation and Fig. 5 demonstrates that it scales significantly better with the number of dimensions. While the scaling seen is not completely independent of the number of dimensions, the scaling observed is much less severe than that seen in the interpolation-based methods.

B. On-The-Fly Memory Mode

Fig. 6 shows that the on-the-fly memory mode marginally increases the matrix-vector product time, but significantly decreases the $\mathcal{H}^2$ matrix construction time. This makes on-the-fly memory ideal for cases where the number of matrix-vector products for each construction is small, while the normal memory mode might be preferred in cases where many matrix-vector products are preformed for each construction.

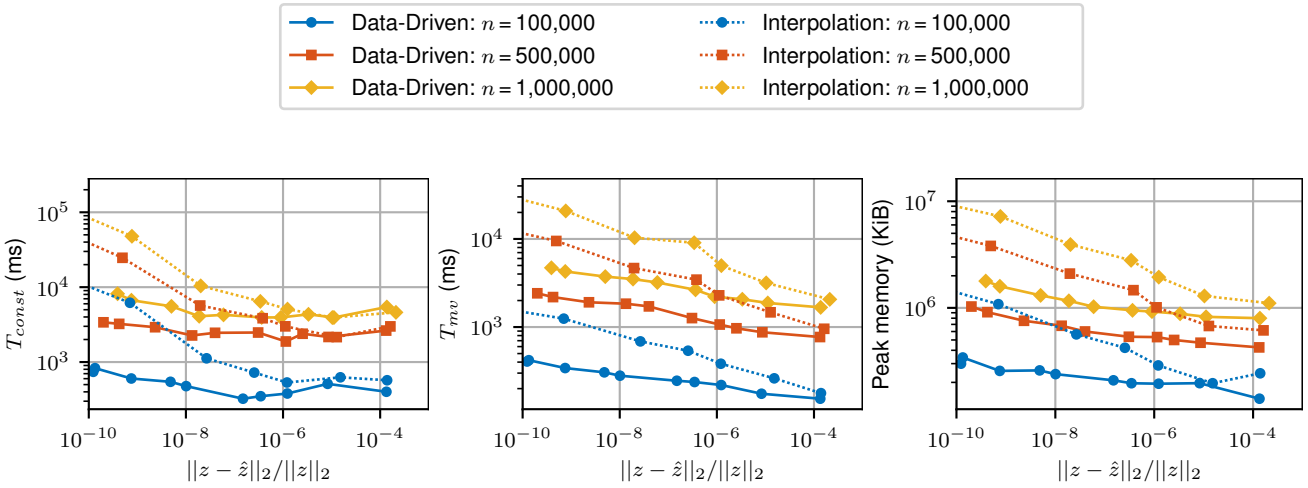
VII. Related Work

There exist a number of packages which, among other features, aim to extend hierarchical and FMM methods to higher dimensions. The STRUctured Matrices PACKage (STRUMPACK) [13] is a distributed memory package based on the HSS matrix format. It requires users to provide a fast matrix-vector multiplication routine in order to use randomized algorithms to perform low-rank compression. ASKIT [27] is a distributed memory package designed for performing high-dimensional kernel summations. It is based on using approximate nearest neighbor information to factorize off-diagonal blocks of kernel matrices. The Geometry-Oblivious FMM (GOFMM) distributed memory package [12] constructs an $\mathcal{H}$ matrix by sampling matrix entries without requiring



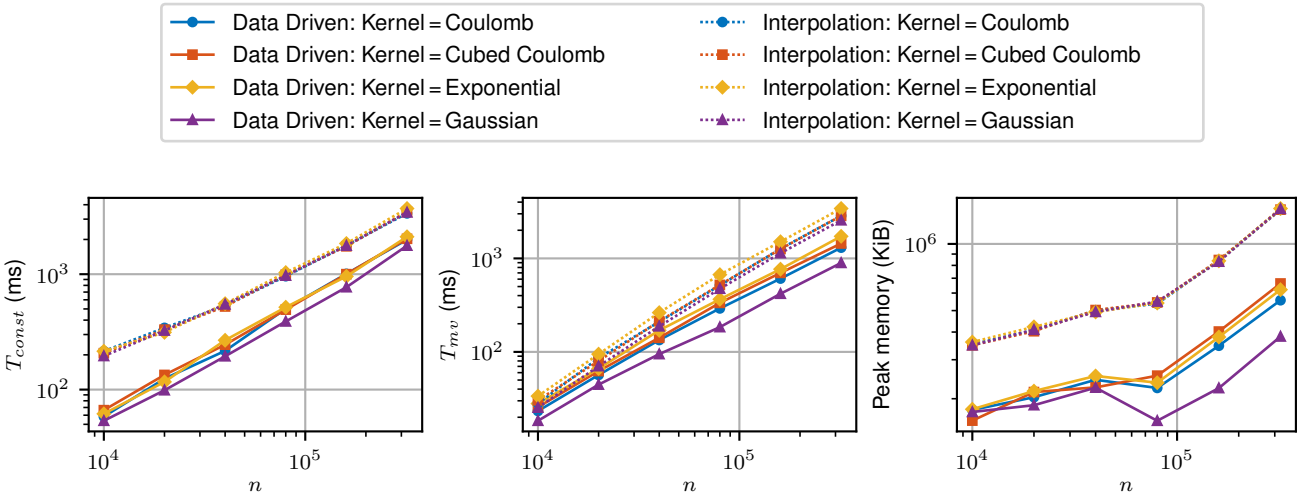
(a) Construction time (ms) (b) Matrix-vector product time (ms) (c) Peak memory usage

Fig. 7. The data-driven and interpolation-based methods vs. thread count. The on-the-fly mode was used for the Coulomb kernel with points in the `cube` distribution, where the test problem has 1,000,000 points and the relative accuracy is fixed around 1e-8.



(a) Construction time (ms) (b) Matrix-vector product time (ms) (c) Peak memory usage

Fig. 8. Data-driven and interpolation-based methods using the on-the-fly memory mode as a function of accuracy for the Coulomb kernel with points in the `cube` distribution.



(a) Construction time (ms) (b) Matrix-vector product time (ms) (c) Peak memory usage

Fig. 9. Data-driven and interpolation-based methods for different kernel functions where the relative accuracy is fixed around 1e-8 for points in the `cube` distribution.

any knowledge of the point coordinates or kernel functions. The main difference between the data-driven method proposed in this paper and these other methods is that the sampling technique in the data-driven method does not require any evaluations or entries of the kernel $\mathcal{K}$ and is performed hierarchically in order to ensure the nested basis property for the $\mathcal{H}^2$ matrix construction.

Meanwhile, many algebraic methods have also been proposed to compress low-rank matrices. Adaptive cross approximation (ACA) [28] can provide compression algebraically using only a few entries of the matrix. However, ACA may fail for general kernel functions and complex geometries due to the heuristic nature of the method. The hybrid cross approximation improves the efficiency of ACA while achieving the convergence seen with interpolation [7]. The CUR decomposition, and the closely related interpolative decomposition, provide a decomposition of the original matrix using a subset of the rows and columns [22] [29]. While interpolative decomposition can be used efficiently in constructing nested bases once candidate bases are determined, its asymptotic complexity makes it infeasible to use to select sample points.

VIII. Conclusion

We demonstrate that bottlenecks associated with hierarchical matrices can be alleviated using our new data-driven and on-the-fly methods. We show that the data-driven method provides an equally general, but computationally more efficient way to calculate generators. Furthermore, the on-the-fly technique allows the memory savings that come with hierarchical matrices to be even more pronounced. Our implementation has near linear scaling with the number of threads for matrix-vector products with all the tested problems. Results demonstrate that both of the methods, individually and cumulatively, result in $\mathcal{H}^2$ matrices that scale linearly (as expected) with the number of points for both computation time and memory usage.

Acknowledgements

The authors gratefully acknowledge funding from the National Science Foundation, grant CMMI-1839591.

References

[1] S. Börm and W. Hackbusch, "Data-sparse approximation by adaptive H^2-matrices," *Computing*, vol. 69, pp. 1–35, 2002.

[2] W. Hackbusch, B. Khoromskij, and S. A. Sauter, "On H2-Matrices," in *Lectures on Applied Mathematics*, H.-J. Bungartz, R. H. W. Hoppe, and C. Zenger, Eds. Springer Berlin Heidelberg, 2000, pp. 9–29.

[3] W. Hackbusch, *Hierarchical Matrices: Algorithms and Analysis*, ser. Springer Series in Computational Mathematics. Berlin Heidelberg: Springer-Verlag, 2015.

[4] J. Barnes and P. Hut, "A hierarchical O(N log N) force-calculation algorithm," *Nature*, vol. 324, pp. 446–449, Dec. 1986.

[5] W. Hackbusch and Z. P. Nowak, "On the fast matrix multiplication in the boundary element method by panel clustering," *Numerische Mathematik*, vol. 54, no. 4, pp. 463–491, 1989.

[6] W. Hackbusch and S. Börm, "H2-matrix approximation of integral operators by interpolation," *Applied Numerical Mathematics*, vol. 43, no. 1, pp. 129–143, Oct. 2002.

[7] S. Börm and L. Grasedyck, "Hybrid cross approximation of integral operators," *Numerische Mathematik*, vol. 101, no. 2, pp. 221–249, Aug. 2005.

[8] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, Dec. 1987.

[9] D. Cai, E. Chow, L. Erlandson, Y. Saad, and Y. Xi, "SMASH: Structured matrix approximation by separation and hierarchy," *Numerical Linear Algebra with Applications*, vol. 25, no. 6, p. e2204, 2018.

[10] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, "Fast algorithms for hierarchically semiseparable matrices," *Numerical Linear Algebra with Applications*, vol. 17, no. 6, pp. 953–976, Dec. 2010.

[11] S. Börm, L. Grasedyck, and W. Hackbusch, "Introduction to hierarchical matrices with applications," *Engineering Analysis with Boundary Elements*, vol. 27, no. 5, pp. 405–422, May 2003.

[12] C. D. Yu, S. Reiz, and G. Biros, "Distributed-memory Hierarchical Compression of Dense SPD Matrices," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 15:1–15:15.

[13] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov, "A Distributed-Memory Package for Dense Hierarchically Semi-Separable Matrix Computations Using Randomization," *ACM Trans. Math. Softw.*, vol. 42, no. 4, pp. 27:1–27:35, Jun. 2016.

[14] S. Börm, *Efficient Numerical Methods for Non-Local Operators: H^2-Matrix Compression, Algorithms and Analysis*, ser. EMS Tracts in Mathematics. EMS, 2010, vol. 14.

[15] V. Rokhlin, "Rapid solution of integral equations of classical potential theory," *Journal of Computational Physics*, vol. 60, no. 2, pp. 187–207, 1985.

[16] D. Cai and J. Xia, "A stable and efficient matrix version of the fast multipole method," *preprint*.

[17] G. Biros, L. Ying, and D. Zorin, "A fast solver for the Stokes equations with distributed forces in complex geometries," *Journal of Computational Physics*, vol. 193, no. 1, pp. 317–348, Jan. 2004.

[18] Lexing Ying, G. Biros, D. Zorin, and H. Langston, "A New Parallel Kernel-Independent Fast Multipole Method," in *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, Nov. 2003, pp. 14–14.

[19] W. Chai and D. J., "An $\mathcal{H}^2$-Matrix-Based Integral-Equation Solver of Reduced Complexity and Controlled Accuracy for Solving Electrodynamic Problems," *IEEE Transactions on Antennas and Propagation*, vol. 57, no. 10, pp. 3147–3159, 2009.

[20] W. Fong and E. Darve, "The black-box fast multipole method," *Journal of Computational Physics*, vol. 228, no. 23, pp. 8712–8725, 2009.

[21] C. K. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Advances in neural information processing systems*, 2001, pp. 682–688.

[22] M. W. Mahoney and P. Drineas, "CUR matrix decompositions for improved data analysis," *Proceedings of the National Academy of Sciences*, vol. 106, no. 3, pp. 697–702, Jan. 2009.

[23] A. Gittens and M. W. Mahoney, "Revisiting the Nyström method for improved large-scale machine learning," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 3977–4041, 2016.

[24] K. Zhang, I. W. Tsang, and J. T. Kwok, "Improved Nyström low-rank approximation and error analysis," in *Proceedings of the 25th International Conference on Machine Learning*. ACM, 2008, pp. 1232–1239.

[25] D. Cai, J. G. Nagy, and Y. Xi, "A Matrix Free Nyström Method via Anchor Net," *preprint*.

[26] P. Glira, N. Pfeifer, C. Briese, and C. Ressl, "A Correspondence Framework for ALS Strip Adjustments based on Variants of the ICP Algorithm," *Photogrammetrie - Fernerkundung - Geoinformation*, vol. 2015, Aug. 2015.

[27] W. March, B. Xiao, C. Yu, and G. Biros, "ASKIT: An Efficient, Parallel Library for High-Dimensional Kernel Summations," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, pp. S720–S749, Jan. 2016.

[28] M. Bebendorf, "Approximation of boundary element matrices," *Numerische Mathematik*, vol. 86, no. 4, pp. 565–589, Oct. 2000.

[29] N. Halko, P. Martinsson, and J. Tropp, "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, Jan. 2011.