Project Title And Abstract

Project Title: High-Performance Matrix Factorization and Solvers on GPU–Accelerated Computers, with applications in Machine Learning, Mathematical Physics, and Engineering.

Abstract. This research emphasizes both theoretical computational complexity and practical applications in machine learning, mathematical physics, and engineering. Specifically, dense linear systems in large-scale kernel approximation in machine learning, finite element discretization of boundary integral equations in engineering, and low-rank Schur complements of large sparse matrix factorization often employ a multilevel structure of low-rank off-diagonal blocks. To solve such systems efficiently, this research presents high-performance algorithms that achieve linear time complexity proportionally to the matrix size under fixed ranks while allowing for tunable precision.

Research Statement

Summary

My research focuses on developing high-performance algorithms that leverage novel numerical algorithms and modern computing architectures to efficiently factorize and solve large dense linear systems.

In mathematical terms, consider a large dense system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^N$ is an unknown variable, $\mathbf{b} \in \mathbb{R}^N$ is given, and dense matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ is also given (although not explicitly formed in the algorithms). My goal is then to factorize the matrix $\mathbf{A}$ in equation (**1**) efficiently, and use the decomposition to solve for the solution $\mathbf{x} \in \mathbb{R}^N$ rapidly.

The most exciting part in my research lies in achieving the optimal $O(N)$ complexity, which surpasses the traditional cubic time $O(N^3)$ complexity of matrix factorization, and obtaining high practical performance of the algorithms. Furthermore, the applications of the dense linear systems in equation (**1**) with multilevel structure of low-rank off-diagonal blocks lie within large-scale kernel approximation in machine learning, finite element discretization of boundary integral equations in physics and engineering, and low-rank Schur complements of large sparse matrix factorization.

Research Motivation

Starting from the traditional matrix factorization algorithms, for example, the LU decomposition, which is often used in scientific and engineering applications, they exhibit a cubic time complexity of $O(N^3)$, where N represents the size of the input matrix. This cubic growth in computational cost becomes a bottleneck as datasets and simulations expand, making it increasingly challenging to

achieve timely results. Therefore, novel algorithms with linear time $O(N)$ complexity are motivated by the desire to overcome these limitations, enabling efficient processing of increasingly large-scale problems.

Transitioning to hierarchical matrix factorization and solver, several fast algorithms with theoretical $O(N)$ complexity have been introduced in the past two decades. However, an investigation on achieving practical high performance leveraging existing techniques is urgently needed. This sets an inspiration for my research to develop new algorithms for the settings of GPUs, taking the advantage of massive parallelization.

Additionally, in the Summer of 2022 and the Summer of 2023, I received two National Science Foundation Funded Ph.D. Research Internships to work in two lab rotations where I learned how to utilize the High-Performance Computing clusters, and the GPUs to solve problems in different fields, from Computational Biology to Computer Science, Electrical and Computing Engineering.

However, the acceleration of iterative algorithms based on sparse matrices on GPUs is limited, regardless of the significant computing power of the GPUs. By contrast, my advisor's previous work [1] shows that acceleration of orders of magnitudes can be achieved for factorizing hierarchically off-diagonal low-rank matrices, and the decomposition can further be used to develop a fast solver. Implementing high-performance linear-time algorithms signifies a paradigm shift, revolutionizing the applicability of matrix factorization across diverse scientific domains.

Preliminary Result

The basic idea is reducing the original problem of solving the large dense system of linear equations $\mathbf{Ax} = \mathbf{b}$ in equation (**1**) to solving a (much) smaller dense linear system

$$\mathbf{Cu} = \mathbf{v}, \tag{2}$$

where $\mathbf{C} \in \mathbb{R}^{M \times M}$ with $M \ll N$.

The algorithm is the following. From the given hierarchical matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ in equation (**1**), the columns of $\mathbf{A}$ are then grouped into b boxes (clusters), where each box contains N/b columns, indexing from 0 to $N/b - 1$. Applying the recursive skeletonization algorithm from the leaf level upwards to the root level, at the start of each level, each box goes through an adaptive decomposition algorithm to return the skeleton and redundant indices of the columns based on the user's input for epsilon accuracy. The redundant indices (columns) will further be discarded, as only the remaining skeleton indices carry relevant information within the user's tuned precision. **Figure 1** below shows the visualization of the skeleton versus the redundant indices at the leaf level in a toy example that factorizes a 256×256 hierarchical matrix for the parallel solver.

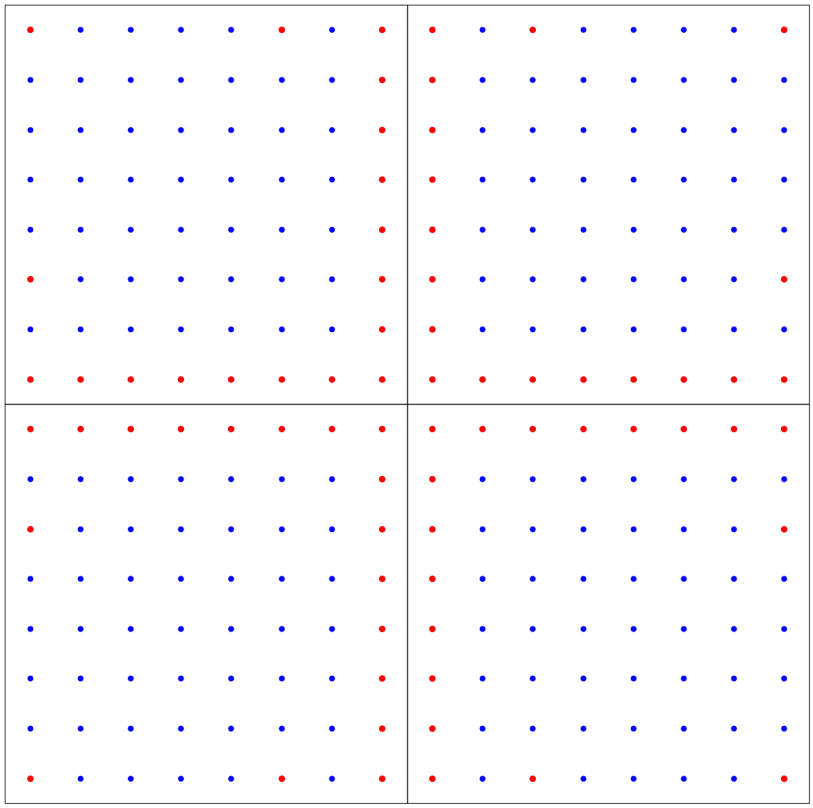

Figure 1: Given a hierarchical matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. After the leaf level, the algorithm returns the skeleton indices (in red) and the redundant indices (in blue). The skeleton indices then carry the relevant information, while the redundant indices do not. Thus, all blue indices can further getting discarded, which results in the problem of solving a linear system $\mathbf{Ax} = \mathbf{b}$ reducing the size nearly by one half, to become a smaller linear system $\mathbf{Cu} = \mathbf{v}$, where $\mathbf{C} \in \mathbb{R}^{\sqrt{N} \times \sqrt{N}}$. The complexity of factorizing the matrix $\mathbb{C}$ is then downsized to $O\left(N^{3/2}\right)$ from the original complexity $O\left(N^3\right)$ of solving the same problem using the matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$.

The algorithm is then used recursively until a small number of M is achieved where the resulting linear system $\mathbf{Cu} = \mathbf{v}$ in equation (**2**) can be factorized and solved within seconds, for example, $M \ll 1024$, given $N \gg 16\,384$. The relationship between $\mathbf{v} \in \mathbb{R}^M$ from equation (**2**) and the original given $\mathbf{b} \in \mathbb{R}^N$ in equation (**1**) then gives the relationship between the solution $\mathbf{u} \in \mathbb{R}^M$ and the original unknown $\mathbf{x} \in \mathbb{R}^N$.

Moving to the subtle difference observed in the CPU threads usage, specifically, 1 thread sequentially versus 3 threads parallely. **Figure 2** below illustrates a toy example that factorizes the hierarchical matrices (through recursive skeletonization) of size $N \times N$ for different values of N, on the CPUs, with 1 thread (sequentially) versus 3 threads (parallely). The factorization is then used for the parallel solver, achieving the user's tunable precision (for example, 10^{-15}).

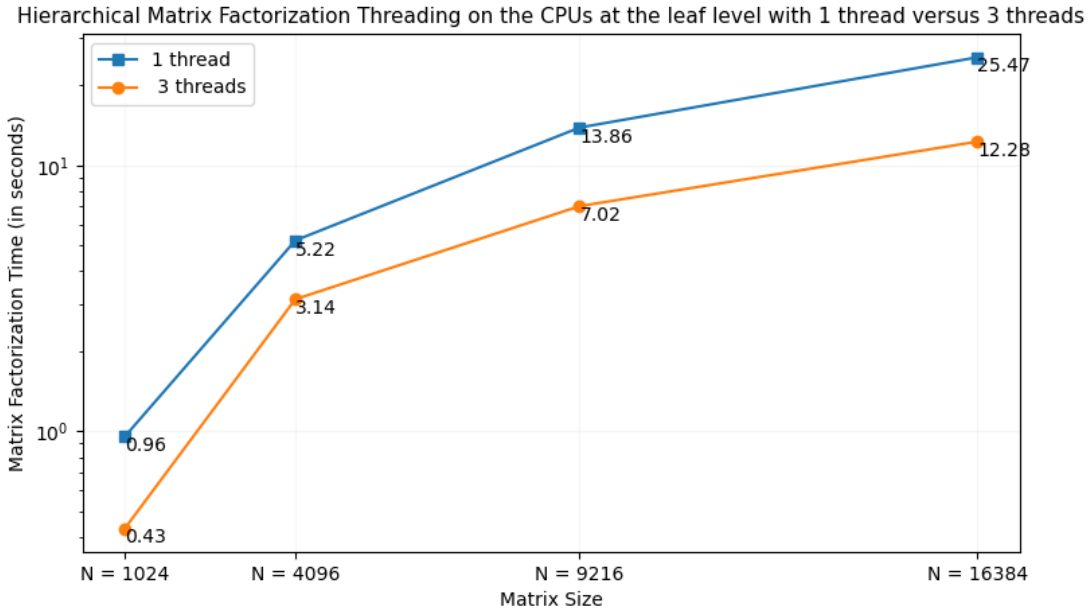

Figure 2: For each $N \in \{1024, 4096, 9216, 16384\}$, the figure shows the amount of time (in seconds) taken to factorize a matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ in the linear system $\mathbf{Ax} = \mathbf{b}$. The decomposition is further used for the parallel solver, specifically through a sequence of matrix-vector multiplication, to achieve the tunable epsilon precision defined by the user, within under 10^{-2} seconds (in this example, the tunable accuracy is 10^{-2} specifically).

What is more important, on an Nvidia V100 GPU with 32 GB of memory, the largest (dense) matrix that can be stored and factorized has size approximately 16 384, and the factorization only takes around 2 seconds. This sets the expectation that parallel processing on GPUs will yield significant improvements in the recursive skeletonization algorithm. Following from the given result, my research goals are directed towards achieving linear time $O(N)$ complexity, thereby extending the impact of research in optimizing computation with hierarchical low-rank structures. Remarkably, my presentation titled "Parallel Recursive Skeletonization Solver for Dense Linear Systems on GPU-Accelerated Computers" was also accepted for an AWM Poster Presentation at the SIAM Annual Meeting (AN24).

Research Plan

The preliminary result from **Figure 2** was generated from my current code using NumPy (with SciPy) and PyTorch, especially with PyTorch Batch Matrix Multiplication. The current NumPy code can then be leveraged to the CuPy code, which is an open-source library for GPU-accelerated computing with Python, and vice versa. Additionally, the Torch Tensors can be moved between the CPUs and the GPUs, with minimal time. Alternatively, I can use North Carolina State University's High-Performance Computing clusters to further implement my factorization algorithms at more larger scales than were previously achievable on the CPUs. Ultimately, my plan is to develop C++ codes to benchmark the performance of these novel high-performance algorithms leveraging modern computing architectures for faster factorizing and solving large dense linear systems of equations, with applications in machine learning, mathematical physics, and engineering.

Furthermore, regarding parallel matrix operations, GPUs, with their multitude of cores, allow for simultaneous execution of matrix operations. This parallelism enables a significant reduction in the time required for factorization. In specific, starting with b, which is the number of blocks (clusters), and p, which is the number of columns of matrix $\mathbf{A}$ in one box, the relationship yields $p = \frac{N}{b}$, where N is the size of the input matrix.

Finally, below is the analysis of time and space complexity for matrix factorization and solver through block skeletonization.

- For the matrix factorization,
  - For the adaptive decomposition to get the redundant and skeleton indices, the time complexity is $\mathcal{O}\left((p^{1/2}b)^3\right)$ and the space complexity is $\mathcal{O}(p^2)$.
  - For the matrix factorization of each block independently, the time complexity is $\mathcal{O}(Np^2b)$ and the space complexity is $\mathcal{O}(N^2 + bp^2)$.
- For the skeletonization solver,
  - For multiplying the decomposition by a vector in $\mathbb{R}^N$, which is a sequence of matrix-vector multiplication, the time complexity is $\mathcal{O}(bp^2)$ and the space complexity is $\mathcal{O}(N)$.

References Cited

[1] C. Chen and P.-G. Martinsson, "Solving linear systems on a gpu with hierarchically off-diagonal low-rank approximations," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. IEEE Press, 2022.