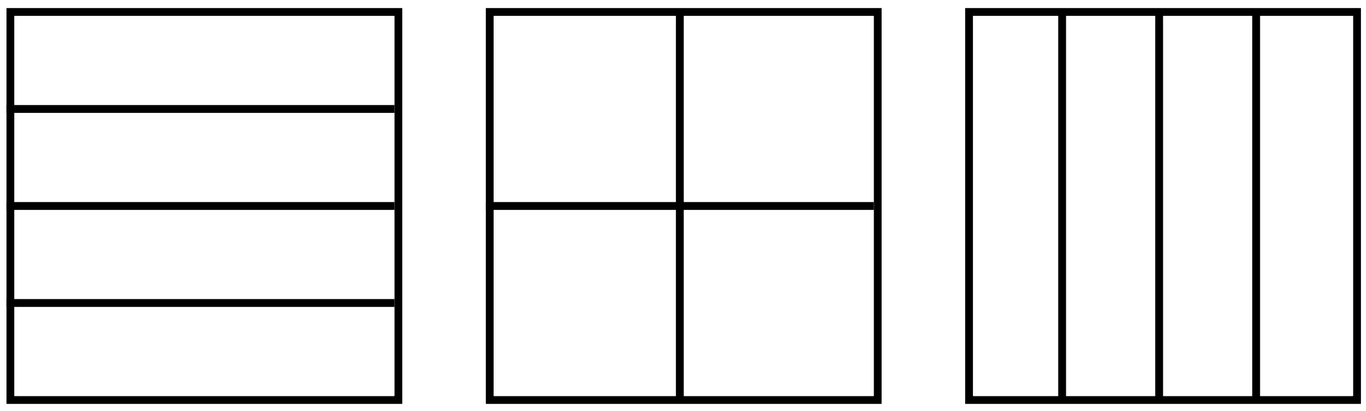
Tensor Butterfly Factorization

Paul Michael Kielstra (UC Berkeley), Yang Liu (LBL), Tianyi Shi (LBL), Hengrui Luo (Rice), Jianliang Qian (MSU)

Lawrence Berkeley Lab

Complementary Low-Rank Matrices and Tensors



An $N \times N$ matrix has the **complementary low-rank (CLR) property** if the rank of any contiguous submatrix with $\leq cN$ elements for some fixed $c \ll N$ is bounded above by some fixed r. These are common in **matrix kernel equations**, which discretize integral equations of the form

$$\int_\Omega K(x,y)\sigma(y)dy = f(x).$$

In particular, the discrete Fourier transform and the Green's function

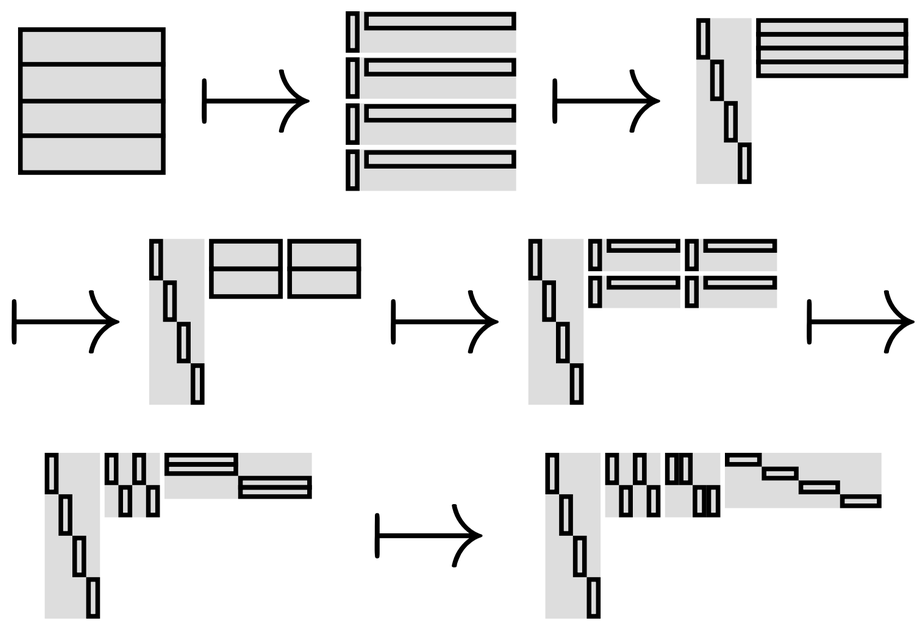
$$K(x,y) = \frac{e^{-\lambda i|x-y|}}{|x-y|},$$

as well as other highly oscillatory operators, exhibit this property.

CLR matrices can be compressed with **butterfly compression**, making it easier to solve matrix kernel equations. The basic idea is to start by splitting the matrix into horizontal (or vertical) blocks, compress them individually, then recursively merge and split the results until we have compressed all possible sets of low-rank submatrices. Often, though, Ω is a high-dimensional space, and the natural discretization of K over a grid is a tensor rather than a matrix. The CLR property for tensors is equivalent to that for matrices, bounding the rank of the highest-rank unfolding matrix of any given small subtensor. Our work extends butterfly compression from matrices to tensors.

Upward and Downward Factorization

The diagram shows an **upward, or row-wise, butterfly factorization** carried out over several steps: (1) a matrix, partitioned into four horizontal blocks; (2) the same matrix, but with each of those blocks individually factored; (3) the factors of the four blocks reassembled into a factorization of the whole matrix, including a smaller vertically-stacked block matrix; (4) the same, but with the block matrix merged horizontally and split vertically; (5) each of the resulting blocks independently factored; (6) the factors of the four blocks reassembled into a factorization of the whole matrix, including a block-diagonal matrix with two blocks; (7) the result of recursively factoring each of the diagonal blocks.

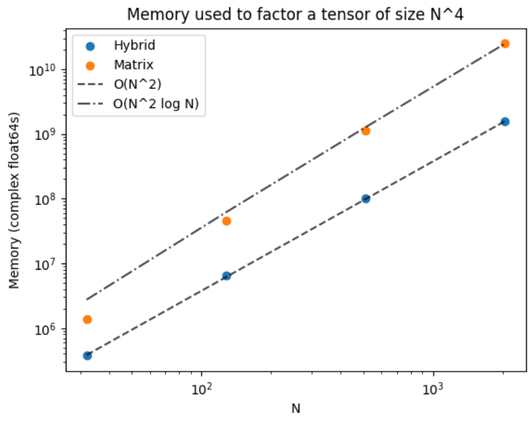


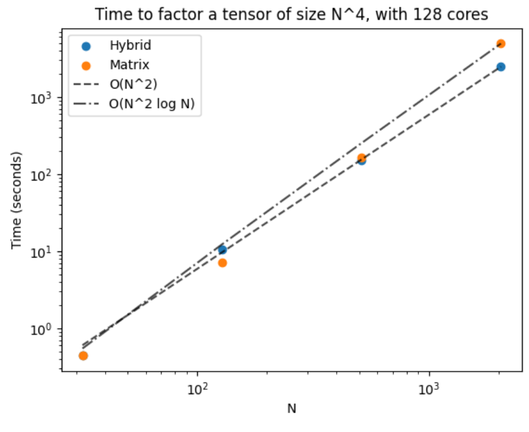
The result requires storing $O(\log N)$ matrices, each of which is sparse and has only $O(N)$ non-zero elements, giving us $O(N \log N)$ non-zero elements in total and allowing much faster matvec operations than the naive $O(N^2)$.

We note that the resulting factorization has an inherent tree structure. The first step, the root node of the tree, requires all the columns to factor. However, after the first step, we split the columns into two subsets, and all later factoring is carried out using either one subset or the other – not both. Even the structure of the matrices with the vertical blocks (the first three of the final four matrices) can be understood from this tree perspective: each node on the tree is associated with some matrices, which are put in a block-diagonal arrangement, and adjacent nodes then have their associated sparse block-diagonal matrices placed next to each other.

It is possible to rewrite the butterfly factorization algorithms for compressing matrices and carrying out matvecs by compressed matrices solely in terms of the tree representation. This eliminates the one part of the algorithm that does not generalize to tensors: that of stacking the submatrices in the right order to allow for the final factorization to be expressed as a product of large sparse matrices. The remainder of the algorithm generalizes immediately to tensors, with one tree being built for every pair of tensor dimensions. The downside is that our factorization cannot be written explicitly as a tensor network.

Results





We factor the tensor defined by discretizing the Green's function above, with $\Omega = [0,1]^2 \times \{0\}$ and $x \in [0,1]^2 \times \{1\}$. In other words, we represent the interactions between parallel two square plates held a distance 1 apart. We take $\lambda = \pi N$, where N is the number of points that we use for our discretization along either side of either plate. The growth order $\lambda \sim N$ ensures that the operator represented by the tensor remains oscillatory overall as N grows. We compare the scaling of our tensor factorization with the current practice of folding the tensor into a matrix and factoring that with matrix butterfly factorization, and see an improvement (including a better order of growth by a factor of log N) both in terms of memory used and in terms of time taken to factor.

These results are also borne out with tensors representing more complicated shapes, such as two adjacent cubes (a 6-tensor rather than the 4-tensor seen here).

References

Cheng, H. et al. "On the Compression of Low Rank Matrices". en. In: *SIAM Journal on Scientific Computing* 26.4 (2005), 1389–1404. ISSN: 1064-8275, 1095-7197.

Liu, Yang et al. "Butterfly Factorization Via Randomized Matrix-Vector Multiplications". en. In: *SIAM Journal on Scientific Computing* 43.2 (2021), A883–A907. ISSN: 1064-8275, 1095-7197.