A Distributed-Memory Fast Multipole Method for Volume Potentials

Dhairya Malhotra
The University of Texas at Austin,
Austin, TX 78712
dhairya.malhotra@gmail.com

George Biros (Advisor)
The University of Texas at Austin,
Austin, TX 78712
gbiros@acm.org

The solution of a constant-coefficient elliptic partial differential equation (PDE) can be computed using an integral transform: a convolution with the fundamental solution of the PDE, also known as a volume potential. We present a Fast Multipole Method (FMM) for efficient computation of volume potentials and use it to construct spatially-adaptive solvers for the Poisson, Stokes and Helmholtz problems. We use high-order piecewise Chebyshev polynomials and an adaptive octree data structure to represent the input and output fields. As with conventional FMM for particle interactions, we evaluate near-field and far-field interactions separately. Near interactions must be computed directly, however this requires evaluating singular and near-singular integrals instead of simple summation for particle interactions. Far-field interactions are evaluated by using kernel independent FMM (KIFMM) by suitably adapting for volume source densities.

Intra-Node Parallelism

We present several optimizations to improve performance on the heterogeneous architecture of Stampede supercomputer. Since accelerators (Xeon Phi) are better suited for tasks with high arithmetic intensity, we evaluate near-field interactions on the accelerator and far-field interactions on the CPU. Computation on Phi and memory transfers to and from the Phi are asynchronous and overlapped with computation on the CPU.

We present a novel scheme for evaluating far-field interactions in KIFMM by using spatial locality of interacting octants to better utilize CPU cache at different levels. The Hadamard product computation in M2L translation of KIFMM has low arithmetic intensity and is therefore memory bound. We interleave data for sibling octants and represent interactions between sibling groups as a stack of 8x8 matrix-vector products. We then combine several interactions along the same direction, evaluated as a stack of matrix-matrix products which we vectorize using SSE and AVX vector intrinsics. We evaluate these interactions in blocks of octants sorted by their MortonId, so that source and target vectors fit in the cache and are reused when computing interactions in the next direction.

Near-interactions in volume FMM are evaluated using precomputed quadratures, since evaluating singular and near-singular integrals on the fly would be extremely costly. Several near interaction are combined to replace matrix-vector products with matrix-matrix products evaluated using DGEMMs. We further use spatial symmetry of interactions to reduce memory footprint and bandwidth requirement for the precomputed quadratures. Interactions in the same symmetry class are related through permutation and scaling operations on the rows and columns of the interaction matrices. We use this to combine interactions in the same symmetry class in to a single matrix-matrix product, thereby improving performance especially for small problem sizes, and significantly improving strong scalability.

The optimizations discussed here resulted in a 7.6× speedup over the unoptimized CPU-only implementation and achieve over 595 Gflop/s of double precision performance on a single node of Stampede.

Distributed Memory Parallelism

For distributed memory parallelism, we use a linear octree, with leaf octants sorted by MortonId, partitioned across processors. Global 2:1 balanced octree is obtained by applying 2:1 balance constraint on each local octree and then globally sorting leaf octants and removing duplicates. The evaluation phase involves a multipole reduction and a hypercube broadcast scheme for ghost octants to construct a local essential tree. The downward pass then proceeds a usual on each local essential tree.

For our largest run on Titan we achieved 567 Tflop/s for a Laplace problem with 74 billion unknowns on a highly non-uniform octree with 26 levels of refinement, solved in 7.8s on 16k compute nodes. On Stampede, we achieved 315 Tflop/s for a Helmholtz problem on 1k compute nodes.

We achieve excellent strong scalability for Laplace kernel up to 256 compute nodes, solving a problem with 124 million unknowns in 0.2s at about 41 Tflop/s. For Helmholtz problems, however we get poor strong scalability and this is a limitation of our scheme.