Accelerating Matrix-Vector Multiplication on Hierarchical Matrices Using Graphical Processing Units

W. Boukaram[†], H. Ltaief[†], [1]A. Litvinenko[‡†], A. Abdelfattah[†] and D. Keyes[†]

[†‡] SRI Uncertainty Quantification Center, [†]Extreme Computing Research Center, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

1 Introduction

Large dense matrices arise from the discretization of many physical phenomena in computational sciences. In statistics very large dense covariance matrices are used for describing random fields and processes. One can, for instance, describe distribution of dust particles in the atmosphere, concentration of mineral resources in the earth's crust or uncertain permeability coefficient in reservoir modeling. When the problem size grows, storing and computing with the full dense matrix becomes prohibitively expensive both in terms of computational complexity and physical memory requirements. Fortunately, these matrices can often be approximated by a class of data sparse matrices called hierarchical matrices ($\mathcal{H}$-matrices) where various sub-blocks of the matrix are approximated by low rank matrices. These matrices can be stored in memory that grows linearly with the problem size. In addition, arithmetic operations on these $\mathcal{H}$-matrices, such as matrix-vector multiplication, can be completed in almost linear time. Originally the $\mathcal{H}$-matrix technique was developed for the approximation of stiffness matrices coming from partial differential and integral equations [4]. Parallelizing these arithmetic operations on the GPU has been the focus of this work and we will present work done on the matrix vector operation on the GPU using the KSPARSE library [1].

A general dense matrix $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ requires $\mathcal{O}(n^2)$ units of memory for the storage and $\mathcal{O}(n^2)$ FLOPS for the matrix-vector multiplication. The $\mathcal{H}$-matrix technique [4, 6, 10] is nothing but a hierarchical division of a given matrix into sub-blocks and further approximation of the majority of them by low-rank matrices (Fig. 1). To define which sub-blocks can be approximated well by low-rank matrices and which not, a so-called admissibility condition is used [4]. When decomposition into sub-blocks is done an important question is, how to compute the low-rank approximations. For this purpose we offer to use the ACA algorithm (or its modifications) [3] which does the approximation job with a linear complexity.
The design of standard $\mathcal{H}$-matrices and its ingredients did not take into account the architecture of the parallel machine, it was not optimized for the efficient, gut scalable $\mathcal{H}$-matrix arithmetics. The optimal choice of admissibility condition (i.e. block partition), the minimal leaf size and the $\mathcal{H}$-matrix rank is very crucial for an efficient parallel implementation.

Various admissibility conditions result in various block partitioning. We are looking for a partitioning which is more appropriate for the graphical Processor Units (GPUs) parallel architecture. To keep works on each processor balanced we are also trying to find an optimal relation between the minimal leaf size and the $\mathcal{H}$-matrix rank.

The parallel $\mathcal{H}$-matrix library HLIB-Pro was implemented by Ronald Kriemann (MIS MPG Leipzig, Germany) [7, 8, 9]. A special focus of HLIBpro lies in the parallelisation

[1]Corresponding author: alexander.litvinenko@kaust.edu.sa, Building 1, Office No 4203, Level 4, 4700 KAUST, Thuwal 23955-6900, Saudi Arabia, Phone: +966-12-8080673

of $\mathcal{H}$-matrix ingredients to shared (threads) and distributed memory machines (MPI). This allows the treatment of very large problems on todays workstations with a multiple-core architecture or parallel clusters with a fast interconnect. HLIBpro uses BLAS/LAPACK for all low-level algorithms.

Our aim is to implement $\mathcal{H}$-matrix techniques on GPUs. The first work for accelerating scientific applications using high performance dense and sparse linear algebra on GPU was already done in [1]. We will also use the MAGMA package [2].

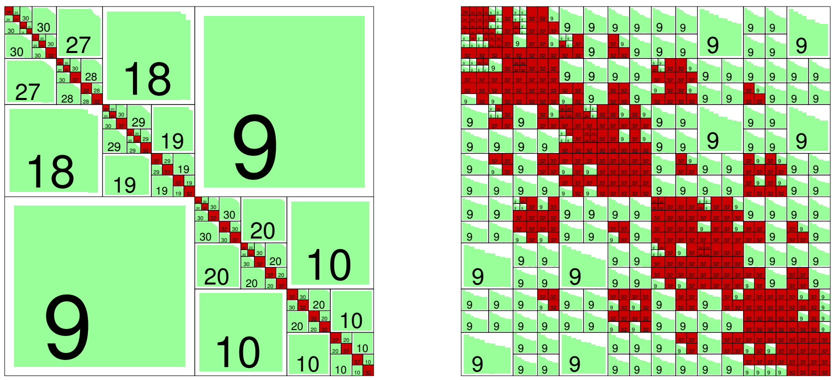

Figure 1: **Two $\mathcal{H}$-matrix approximations of the Matérn covariance matrix with the so-called (left) weak and (right) strong admissibility conditions. The green blocks denote low-rank matrices (the number inside denotes the rank) and the red blocks denote dense matrices. Steps in green blocks denote the eigenvalue decay in log-scale.**

1.1 Matérn covariance functions

The aim of our work is an $\mathcal{H}$-matrix approximation of Matérn covariance functions on GPU. Among many available covariance models, the Matérn family has gained widespread interest in recent years. The class of Matérn covariance functions [Matérn (1960), (Handcock and Stein, 1993 and Stein, 1999)] is very widely used in spatial statistics, geostatistics, machine learning, image analysis, and other areas. It is e.g. commonly used to define the statistical covariance between measurements made at two points (the distance between two points is denoted by r). The Matérn class of covariance functions is defined as

$$C(r) := C_{\nu,L_0}(r) = \frac{2\sigma^2}{\Gamma(\nu)}\left(\frac{r}{2L_0}\right)^{\nu} K_{\nu}\left(\frac{r}{L_0}\right), \tag{1}$$

where $\nu > 0$ depends upon the smoothness of the random field, with larger values of ν corresponding to smoother fields; and $L_0 > 0$ is a spatial range parameter that measures how quickly the correlation of the random field decays with distance, with larger L_0 corresponding to a slower decay (keeping ν fixed). When $\nu = 1/2$, the Matérn covariance function reduces to the exponential covariance model and describes a rough field. The value $\nu = \infty$ corresponds to a Gaussian covariance model which describes a very smooth field, in fact a field which is

infinitely differentiable. Sample functions from Matérn forms are $\lfloor \nu - 1 \rfloor$ times differentiable. Thus, the hyperparameter ν can control the degree of smoothness.

Further applications of covariance matrices are: Kriging estimate (includes matrix vector multiplication), computation of the conditional covariance, estimation of the variance (computation of the diagonal of the conditional covariance matrix), computation of the Cholesky decomposition (used, for instance, for random fields generation) and geostatistical optimal design [11].

2 Graphical Processing Units and CUDA

Graphical Processing Units (GPUs) are popular high performance hardware accelerators in modern supercomputers. GPU programming has a different model than that for CPUs, which means that many numerical kernels have to be redesigned and optimized specifically for this architecture. GPUs usually outperform multicore CPUs in some compute intensive and massively parallel applications that have regular processing patterns. However, most scientific applications rely on crucial memory-bound kernels and may witness bottlenecks due to the overhead of the memory bus latency. They can still take advantage of the GPU compute power capabilities, provided that an efficient architecture-aware design is achieved. In [1] the author presented a uniform design strategy for optimizing critical memory-bound kernels on GPUs. Based on hierarchical register blocking, double buffering and latency hiding techniques, this strategy leverages the performance of a wide range of standard numerical kernels found in dense and sparse linear algebra libraries. The author focuses on matrix-vector multiplication kernels (MVM) as representative and most important memory-bound operations in this context. Each kernel inherits the benefits of the proposed strategies. By exposing a proper set of tuning parameters, the strategy is flexible enough to suit different types of matrices, ranging from large dense matrices, to sparse matrices with dense block structures, while high performance is maintained. Furthermore, the tuning parameters are used to maintain the relative performance across different GPU architectures.

References

[1] Ahmad, Ahmad Mohammad, *Accelerating Scientific Applications using High Performance Dense and Sparse Linear Algebra Kernels on GPUs*, Dissertation, KAUST, Saudi Arabia, http://repository.kaust.edu.sa/kaust/handle/10754/346955, (2015)

[2] Agullo, E. et al.,, *Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects*, J. of Ph. 180 (1), (2009).

[3] Bebendorf, M. and Rjasanow, S., *Adaptive low-rank approximation of collocation matrices*, Computing 70 (1), pp 1–24, (2003).

[4] Hackbusch, W., *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices*, Computing 62 (2), pp 89–108, (1999).

[5] Izadi M., *Hierarchical matrix techniques on massively parallel computers*, Dissertation, University Leipzig, Germany, (2012).

[6] Khoromskij, B. N., Litvinenko, A. and Matthies, H. G, *Application of hierarchical matrices for computing the Karhunen–Loève expansion*, Computing 84, (1-2), pp 49–67, (2009).

[7] Kriemann, R., *Parallele algorithmen fuer $\mathcal{H}$-Matrizen*, Dissertation, University of Kiel, http://e-diss.uni-kiel.de/diss_1511/index.htm, (2005).

[8] Kriemann, R., *$\mathcal{H}$-LU Factorization on Many-Core Systems*, MPI Leipzig Preprint 5, accepted by Computing and Visualization in Science, (2015).

[9] Kriemann, R., *Parallel $\mathcal{H}$-Matrix Arithmetics on Shared Memory Systems*, Computing 74, pp 273–297, (2005).

[10] Litvinenko, A., *Application of hierarchical matrices for solving multiscale problems*, Dissertation, Leipzig University, Germany, http://www.wire.tu-bs.de/mitarbeiter/litvinen/diss.pdf, (2006).

[11] Nowak, W., Litvinenko, A., *Kriging and Spatial Design Accelerated by Orders of Magnitude: Combining Low-Rank Covariance Approximations with FFT-Techniques*, Mathematical Geosciences 45 (4), 411-435, (2013).