

Tile Low Rank Cholesky Factorization for Climate/Weather Modeling Applications on Manycore Architectures

Kadir Akbudak, Hatem Ltaief^(✉), Aleksandr Mikhalev, and David Keyes

Extreme Computing Research Center, Division of Computer,
Electrical, and Mathematical Sciences and Engineering,
King Abdullah University of Science and Technology,
Thuwal, Kingdom of Saudi Arabia

{Kadir.Akbudak,Hatem.Ltaief,Aleksandr.Mikhalev,David.Keyes}@kaust.edu.sa

Abstract. Covariance matrices are ubiquitous in computational science and engineering. In particular, large covariance matrices arise from multivariate spatial data sets, for instance, in climate/weather modeling applications to improve prediction using statistical methods and spatial data. One of the most time-consuming computational steps consists in calculating the Cholesky factorization of the symmetric, positive-definite covariance matrix problem. The structure of such covariance matrices is also often data-sparse, in other words, effectively of low rank, though formally dense. While not typically globally of low rank, covariance matrices in which correlation decays with distance are nearly always hierarchically of low rank. While symmetry and positive definiteness should be, and nearly always are, exploited for performance purposes, exploiting low rank character in this context is very recent, and will be a key to solving these challenging problems at large-scale dimensions. The authors design a new and flexible tile row rank Cholesky factorization and propose a high performance implementation using OpenMP task-based programming model on various leading-edge manycore architectures. Performance comparisons and memory footprint saving on up to $200K \times 200K$ covariance matrix size show a gain of more than an order of magnitude for both metrics, against state-of-the-art open-source and vendor optimized numerical libraries, while preserving the numerical accuracy fidelity of the original model. This research represents an important milestone in enabling large-scale simulations for covariance-based scientific applications.

1 Introduction

The march toward exascale computing is well, underway with today's fastest systems capable of achieving near 100 PFlop/s in sustained peak performance on million of cores [22]. Technology scaling with incremental hardware evolution will most probably enable to cross the exascale barrier by 2021, as recently announced by the US Department of Energy. However, the current hardware roadmap development will not be able to get to exascale within a power budget of 20 MW that

many hardware architects and research agencies consider as a practical upper limit for such a system. Although, this power gap may be further reduced with advanced energy-efficient devices (e.g., hardware accelerators), algorithmic novelties around synchronization-reducing and communication-reducing concepts are paramount not only to ultimately design an exascale system at reasonable power levels, but also to ensure an efficient usage of the massively parallel underlying hardware.

Covariance matrices are ubiquitous in computational science and engineering. Large covariance matrices arise from multivariate spatial data sets, for instance, in seismic inversion to obtain estimates of uncertainty [11], in computational ground-based astronomy to enhance the observed image quality by filtering out the noise coming from the adaptive optics instrumentation and the atmospheric turbulence [21], or in climate/weather modeling to improve prediction using geospatial statistics approaches [24]. All the aforementioned scientific applications boil down to calculating the Cholesky factorization of a symmetric, positive-definite matrix problem, which turns out to be the most time consuming computational phase in their various respective simulations. The structure of these covariance matrices is often data-sparse, in other words, effectively of low rank, though apparently dense. The dense Cholesky allows to perform an exact factorization up to the machine precision while the low rank variant produces an approximation of the Cholesky factor up to a desired accuracy threshold. While not necessarily globally of low rank, covariance matrices in which correlation decays with distance are nearly always hierarchically of low rank. While symmetry and positive definiteness should be, and nearly always are, exploited for performance purposes, exploiting low rank character in this context is very recent, and will be a key to solving these challenging problems. Because these low rank approximations [14] operate on a lossy (but controllable) compressed representation of the original dense data structure, this directly translates into lower arithmetic complexities and memory footprint saving, which are key elements for reducing data movement and time to solution, while staying within the future exascale system power envelope.

Fully dense linear algebra approaches encounter high ($O(N^3)$) arithmetic complexity and the overhead of a large ($O(N^2)$) memory footprint where N is the number of objects to be correlated. This scaling is impractical when dealing with large data sets which, today, could usefully translate into covariance matrices with N in the billions. To tackle such problems, we study the numerical accuracy, the memory footprint and the performance of the tile low rank Cholesky factorization (TLR Cholesky) in the context of climate/weather modeling applications [24], by exploiting the data sparsity in the covariance matrix and relying on task-based programming model for asynchrony and dynamic load balancing. Experiments are conducted on Intel Xeon Haswell/Broadwell and Intel Xeon Phi Knights Landing. Results reported indicate up to an order of magnitude of memory saving as well as time to solution reduction on $200K \times 200K$ covariance matrix size, compared to the native dense Cholesky factorization, as implemented in the state-of-the-art high performance open-source and vendor software libraries. This emerging family of low rank matrix computations

represents a breakthrough for the statistical computing community, for which the default use of high-level simulation software tool such as *R* [1] may often be limited in dimension scaling by expensive dense linear algebra kernels.

The remainder of the paper is organized as follows. Section 2 details related work. Section 3 highlights our contributions. Section 4 describes the climate/weather modeling simulation based on a geospatial statistics approach applied to the covariance matrix. We recall the state-of-the-art dense Cholesky factorization in Sect. 5, which is the most time-consuming phase of the application studied here. Section 6 outlines a new TLR Cholesky factorization, which additionally exploits the data sparsity of the dense covariance matrix. Numerical accuracy is provided in Sect. 7 and shows the flexible and robustness of the TLR matrix approximation for Cholesky factorization. Section 8 gives implementation details of the TLR Cholesky, which relies on the OpenMP task-based programming model for performance purposes. Section 9 presents the performance results and analysis and compares TLR Cholesky factorization against existing state-of-the-art implementations. We conclude in Sect. 10.

2 Related Work

Low rank matrix approximations under the rubric of hierarchical matrices or \mathcal{H} -matrices [14, 17] have been extensively studied in the literature since the end of the 1990's, mainly from a theoretical perspective, with critical bounds derived on algorithmic complexities and memory footprint. Since then, many new data compression formats for \mathcal{H} -matrix approximation have emerged to cover a wide range of scientific applications such as finite and boundary element methods and Gaussian processes. These compression formats, e.g., hierarchically semi-separable (HSS), \mathcal{H}^2 -matrix, hierarchically off-diagonal low-rank (HODLR), block low rank (BLR), are categorized depending on the data format structure (i.e., nested or non-nested basis) and the admissibility condition (i.e., standard/strong or weak). The former impacts both aforementioned bounds while the latter allows a fine-grained capture of the low rank structure of the matrix off-diagonal blocks.

Low rank matrix approximation relying on nested bases (i.e., \mathcal{H}^2 -matrix [9, 10, 15, 16] and HSS [23]) provides the best theoretical bounds for algorithmic complexities and memory footprint for scientific problems which exhibit nested row and column basis. The latter are challenging to implement efficiently on manycore architectures due to synchronization points in the recursive tree. Data compression formats based on non-nested bases (i.e., \mathcal{H} -matrix [18, 20], HODLR [3, 6] and BLR [4]) have higher bounds but they are often capable of handling broader range of scientific applications than low rank data format with nested basis. In particular, BLR is probably the most straightforward low rank approximation format to implement because it does not rely on a recursive tree and adopts a flattened data structure instead, at the expense of showing the highest algorithmic complexity. BLR is currently under investigation in MUMPS [5] during the Schur complement involving frontal matrices [4].

This work presents the tile low rank (TLR) data format, which is similar to BLR, although it takes root from the well-known tile algorithms, as implemented in dense linear algebra libraries such as PLASMA [27]. Last but not least, this work aims at filling in the software gap by providing high performance TLR approximations for matrix operations, and therefore, minimizing the memory and complexity overhead of using dense matrix computations as the native approach.

3 Contributions

The contributions of the paper are fourfold. The authors (1) design a new and flexible tile low rank Cholesky factorization for dense covariance matrices, (2) provide a performance assessment on various leading-edge hardware architectures by looking at numerical accuracy, memory footprint and time to solution, (3) compare TLR Cholesky factorization against state-of-the-art vendor and open-source dense linear algebra libraries such as MKL [19] and PLASMA [27], respectively and (4) leverage performance of emerging architectures for climate/weather modeling applications.

4 Climate/Weather Modeling Applications

Large covariance matrices arise from multivariate spatial data sets in climate/weather modeling simulations to improve prediction using statistical methods and spatial data [24]. The crux of the modeling effort is to estimate a maximum likelihood objective function based on observations, as follows:

$$l(\theta) = -\frac{1}{2} Z^T \Sigma^{-1}(\theta) Z - \frac{1}{2} \log |\Sigma(\theta)|, \quad (1)$$

where θ is the vector of parameters to be tuned, Z a vector of observations, and Σ the covariance matrix, and where the vertical bars indicate a determinant. These matrices are symmetric, positive-definite and are based on covariances of presumed Gaussian processes. If we have only one Gaussian process, then the corresponding covariance matrix is simply scalar. In N -dimensional case, with N being the number of geographical locations to be correlated, we have N Gaussian processes, which leads to square N -by- N matrix. As is apparent from Eq. 1, the computational bottleneck of the maximum likelihood estimation is the calculation of the Cholesky factorization of the dense covariance matrix Σ , which is necessary to solve the linear system (i.e., forward and backward substitutions) as well as getting the logarithm of the covariance matrix determinant (i.e., product of the diagonal elements of the Cholesky factor). The dense Cholesky factorization, for instance, as implemented in the state-of-the-art simulation software *R*, requires $O(N^3)$ operations on $O(N^2)$ data. This is a prohibitive approach, given that N may be in the order of billions in readily contemplated applications. Alternative less expensive approximation approaches exist such as element

thresholding, subsampling and iterative methods, however, these methods sacrifice the fidelity of the underlying statistical model.

The main idea consists in exploiting the data sparsity of the formally dense covariance matrix $\Sigma(\theta)$. This represents a cheaper computational algorithmic design, while still preserving the model fidelity up to a given accuracy. The resulting matrix is hierarchically of low rank and can be compressed using the tile low rank data format, which enables to better capture the low rankness structure of the off-diagonal blocks, thanks to its strong admissibility condition, as opposed to HODLR data format.

In this paper, we synthesize a set of covariance matrices as follows: given an N -by- N uniform grid of particles in unit square with exponential interaction $f(x, y) = e^{-\frac{|x-y|}{\beta}}$, we add random noise to coordinates of each particle and sort them in Morton order. So, if we have set of N^2 particles $\{X_i\}_{i=1}^{N^2}$, each element of the covariance matrix $\Sigma(\theta)$ can be defined as follows:

$$A_{ij} = e^{-\frac{r(X_i, X_j)}{\beta}}, \quad (2)$$

where $r(X_i, X_j)$ is a distance between particles X_i and X_j and β represents a covariance parameter, which measures the correlation between the Gaussian processes. Although the current kernel considered for the matrix generation is the Gaussian kernel, Matérn kernels, for instance, among other Gaussian processes kernels, can also be handled in the same manner. All in all, these kernels are asymptotically smooth, which lead to the possibility of low-rank approximations of different blocks of a matrix [25]. The ranks depend on how the clusterization of the spatial particles occurs, given the relative distance from one cluster to another. It is also noteworthy to mention that, in case of uniform distribution of N spatial points with N power of 2, Morton order space-filling curve may nearly be optimal.

5 State-of-the-Art Dense Cholesky Factorization

This section recalls the algorithmic evolution of the dense Cholesky factorization. The Cholesky factorization of an $N \times N$ real symmetric, positive-definite matrix A has the form $A = LL^T$, where L is an $N \times N$ real lower triangular matrix with positive diagonal elements.

5.1 Block Algorithms

Block algorithms, as implemented in LAPACK [7], emerged with cache-friendly hardware architectures in the late 1990's. The matrix computation is decomposed in two successive phases. The panel factorization consists in applying Level 2 BLAS transformations within a panel of the matrix only, followed by the update of the trailing submatrix, which accumulates all transformations from the current panel and applies them by means of Level 3 BLAS on the unreduced part of the matrix, as depicted in Fig. 1(a). The matrix computation

algorithms proceed then on a smaller subset of the overall matrix as in Fig. 1(b), until the matrix is completely transformed, as seen in Fig. 1(c). Parallel performance is only exploited during the update of the trailing submatrix, during calls to compute-intensive multithreaded Level 3 BLAS, as provided for instance by vendor optimized BLAS implementations (e.g., Intel MKL [19]). Artifactual synchronization points in-between computational phases impede parallel performance, especially in presence of multicore architectures [2].

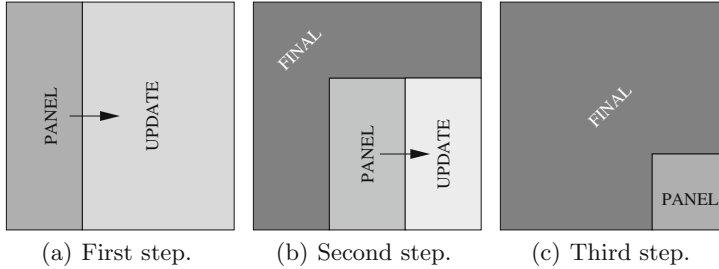


Fig. 1. Block algorithms: LAPACK/MKL.

5.2 Tile Algorithms

Tile algorithms emerged with multicore architectures a decade ago. The dense matrix is broken into tiles, as seen in Fig. 2, where elements are contiguous in memory within each tile. Tile algorithms weaken the synchronization points revealed in block algorithms by bringing the parallelism in multithreaded BLAS to the fore. They create opportunities for asynchronous execution with potentials for look-ahead optimizations. The whole algorithm may be then represented as a directed acyclic graph, where nodes are computational tasks and edges define data dependencies between them. A dynamic runtime system is employed to schedule tasks across processing units, while ensuring data dependencies are not violated. PLASMA [27] and FLAME [13] represent the two state-of-the-art dense linear algebra libraries, which rely on tile algorithmic formulation.

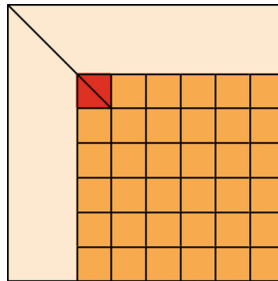


Fig. 2. Tile algorithms: PLASMA/FLAME.

6 The Tile Low Rank Cholesky Factorization

This section presents the tile low rank (TLR) approximation and Cholesky factorization.

The first phase is to create an approximation of each off-diagonal tile, typically by performing a singular value decomposition (SVD) and by keeping only the most significant singular values and their corresponding singular vectors, depending on the selected accuracy. The latter is a parameter, which is often application-specific. The diagonal tiles are typically full rank and cannot be approximated. The obtained off-diagonal data structure is no more a dense tile of contiguous elements but an outer product of two rectangular matrices $U_{ij} \times V_{ij}$ of size $nb \times k$, with nb the tile size and k the matrix rank (i.e., the k most significant singular values/vectors), as shown in Fig. 3. Our current TLR approximation offers two variants. Fixed ranks can be used to apply truncation across all off-diagonal tiles, independently of the data, at the cost of obtaining lower or higher accuracy across the tiles (see Fig. 3(a)). Though seemingly brute force, this may be the most cost-effective and per-iteration performant form of preconditioning for iterative solvers. The fixed accuracy variant permits to smoothly approximate the off-diagonal tiles depending on the accuracy needed by the application. This engenders variable ranks per tiles, as seen in Fig. 3(b), with an arbitrary illustration for six ranks (k_1 to k_6).

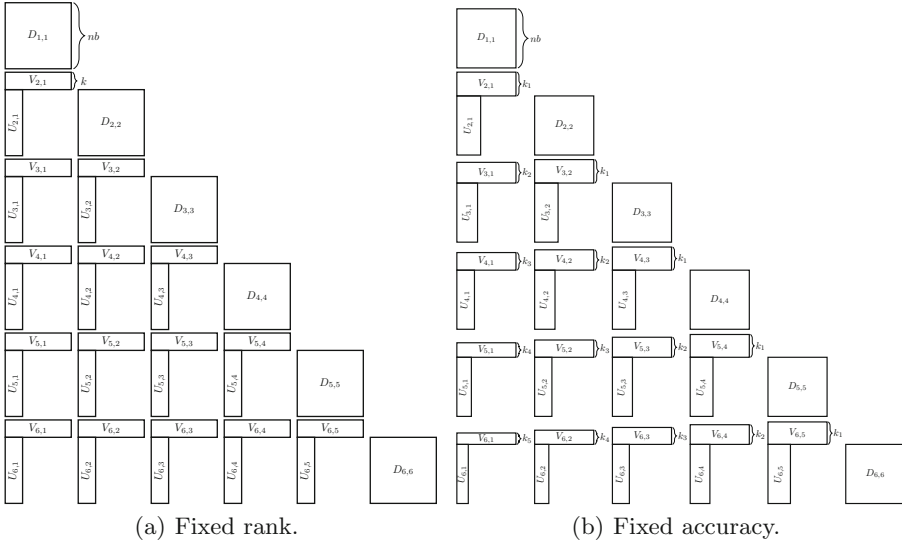


Fig. 3. Tile low rank matrix representation.

Once the dense matrix is approximated by means of tile low rank, a new family of linear algebra algorithms needs to be implemented to take into consideration the new compressed data layout. For the TLR Cholesky factorization,

we reuse some of the ideas developed in the PLASMA library [27], although new monolithic kernels have to be designed. When using fixed rank k , all off-diagonal tiles of size nb are represented by a data structure of identical size, i.e., $nb \times k$. With fixed accuracy, the rank obtained may differ from one tile to another to maintain the expected accuracy threshold. Therefore, load imbalance issues may increase idle time and it is then paramount to rely on dynamic runtime systems in order to mitigate this overhead by ensuring all resources stay busy throughout the matrix computations.

7 Numerical Accuracy

This section aims at highlighting the robustness of the TLR compression and Cholesky factorization. We look first at synthetic covariance matrices and then at real geospatial covariance matrices from climate/weather modeling applications based on Gaussian processes.

7.1 Synthetic Matrices

Synthetic matrices are important to demonstrate the numerical robustness for new matrix algebra software. We create a template diagonal matrix S with three specific singular value decay rates (named as base 2, base 3, base 4), as depicted in Fig. 4. The singular values or diagonal elements of S in descending order follow these decay rates and reach close to machine precision in double precision arithmetic ($1e - 16$) for the first 53, 33 and 26 singular values for base 2, base 3, and base 4, respectively. This matrix S is then multiplied on the left and right sides by orthogonal matrices to generate each data-sparse off-diagonal tiles.

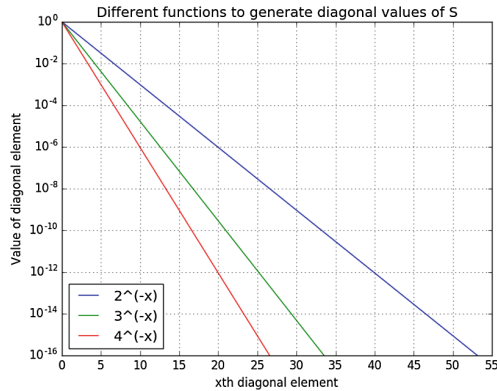


Fig. 4. Singular values decay rates and distribution of the template diagonal matrix.

Once all off-diagonal tiles have been generated, they can be compressed using an SVD. Extensive numerical experiments have been conducted on synthetic

covariance matrices to validate our TLR approach. The heat map Fig. 5 reports the accuracy obtained for various fixed ranks and tile sizes (Fig. 5(a)) using base 2 decay rate and the corresponding digit difference with the full dense Cholesky factorization (Fig. 5(b)). The heat map Fig. 6 reports the accuracy obtained for various fixed ranks and tile sizes (Fig. 6(a)) using base 3 decay rate and the corresponding digit difference with the full dense Cholesky factorization (Fig. 6(b)). The heat map Fig. 7 reports the accuracy obtained for various fixed ranks and tile sizes (Fig. 7(a)) using base 4 decay rate and the corresponding digit difference with the full dense Cholesky factorization (Fig. 7(b)). Indeed, one can notice that double precision arithmetic (10^{-16}) is achieved from rank truncations starting from 53, 33 and 26 singular values for base 2, base 3, and base 4, respectively.

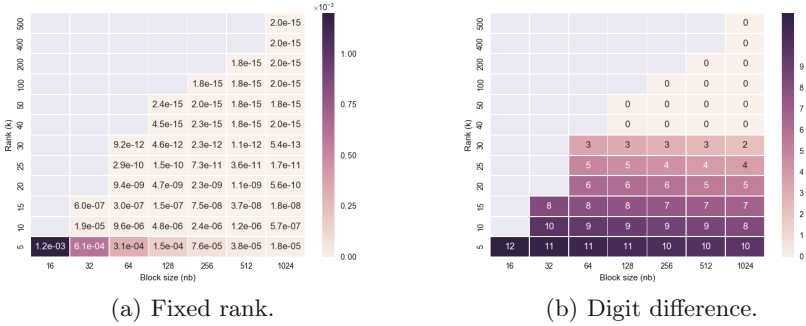


Fig. 5. Singular value distribution base 2.

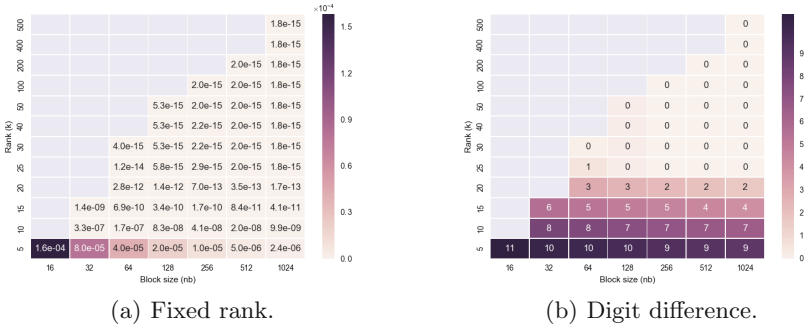


Fig. 6. Singular value distribution base 3.

Figure 8 shows fixed accuracy instead, using base 3 decay rate, and reveals the resulting rank with the following obvious rule: the higher the accuracy needed, the higher the rank. Last but not least, the tile size nb parameter does not really matter for these synthetic matrices in terms of numerical accuracy because the template diagonal tile S is the same one used during data-sparse off-diagonal tile generation. In fact, for the dense Cholesky factorization, the parameter nb

has an impact only on performance as it trades-off concurrency with sequential kernel performance. For tile low rank Cholesky factorization, nb has, in addition, a direct impact on the overall algorithmic complexity. For instance, for a given matrix size N , a large tile size nb would engender small memory footprint as well as number of floating-point operations at the price of a lower concurrency. On the contrary, if the large tile size nb would have been further decomposed into smaller ones, this would engender larger memory footprint as well as number of floating-point operations at the price of a higher concurrency.

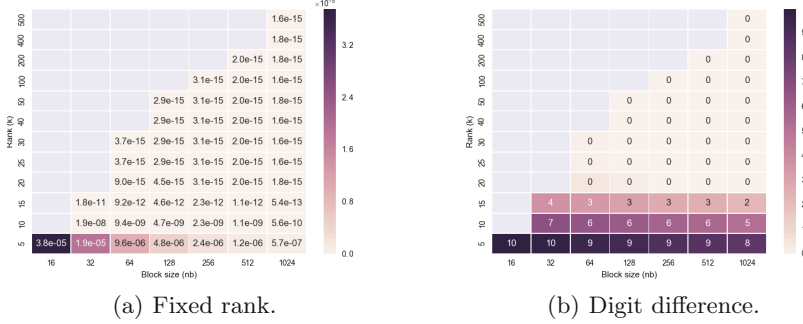


Fig. 7. Singular value distribution base 4.

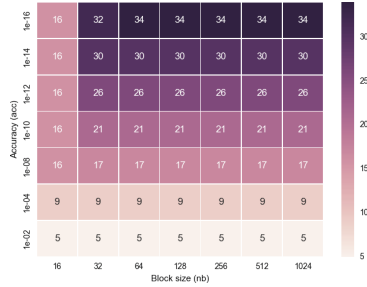


Fig. 8. Fixed accuracy for base 3.

7.2 Geospatial Statistics

The typical accuracy required for the studied climate/weather modeling application is 10^{-9} . Given this accuracy, Fig. 9 highlights the rank distributions for for 16384×16384 covariance matrix generated by Eq. 2 with $\beta = 0.1$ for $nb = 64, 128$ and 256: the whiter the picture is, the greater its data sparsity. The diagonal tiles are full ranks, regardless of the tile size, while the off-diagonal tiles are mostly data-sparse and can be approximated accordingly. In fact, perhaps the most striking feedback about this figure is that the majority of off-diagonal tiles can be dramatically approximated, while the initial matrix is completely dense.

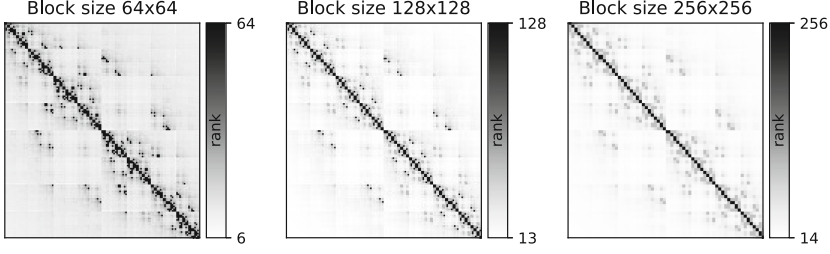
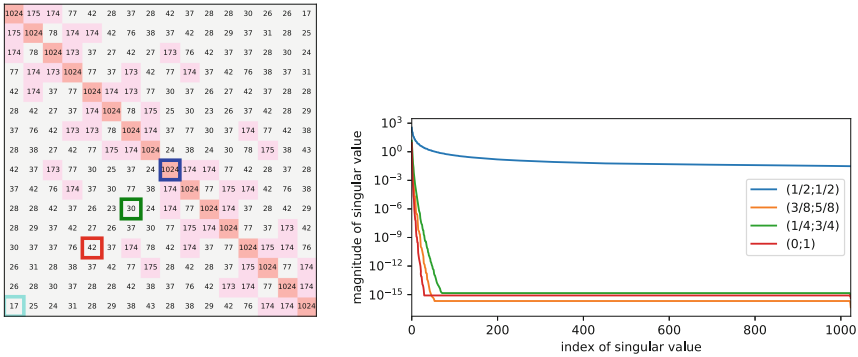


Fig. 9. Rank distributions for 16384×16384 covariance matrix using various tile sizes.

Figure 10 reveals the heat map of the rank and singular value distributions for 16384×16384 covariance matrix using $nb = 1024$. In particular, Fig. 10(a) shows the rank for each tile after using the application-specific accuracy of 10^{-9} . If dense linear algebra approaches were used, the bottom left tile of rank 17 with $U = 1024 \times 17$ and $V = 1024 \times 17$ would have been considered full rank and of size 1024×1024 , instead. And this phenomenon is further exacerbated when looking at Fig. 10(b), which portrays the singular value distributions of selected off-diagonal tiles. While the singular values of the diagonal tiles are all significant, the singular values of off-diagonal tiles are actually characterized by an exponential decay, which has to be exploited for performance and storage purposes. Such characteristic may not be captured by weak admissible data compression formats, such as HODLR and HSS, due to nested dissection which operates only for diagonal blocks. The off-diagonal blocks may then necessitate larger rank to get compressed, which may have a non-negligible impact on performance and memory footprint.



(a) Rank distributions.

(b) Singular values decay of marked tiles in Fig. 10(a).

Fig. 10. Rank and singular value distributions for a 16384×16384 covariance matrix using $nb = 1024$ with an accuracy threshold set to 10^{-9} .

8 High Performance Implementations

This section describes the high performance implementation of the TLR Cholesky factorization.

8.1 Numerical Kernels

The sequential TLR Cholesky algorithm can be expressed with the following four computational kernels:

HCORE_DPOTRF: The kernel performs the Cholesky factorization of a diagonal (lower triangular) tile. It is similar to LAPACK DPOTRF since the diagonal tiles are dense and full rank.

HCORE_DTRSM: The operation applies an update to an off-diagonal low-rank tile of the input matrix, resulting from factorization of the diagonal tile above it and overrides it with the final elements of the output matrix: $V_{(i,k)} = V_{(i,k)} \times D_{(k,k)}^{-1}$. The operation is a triangular solve.

HCORE_DSYRK: The kernel applies updates to a diagonal (lower triangular) tile of the input matrix, resulting from factorization of the low-rank tiles to the left of it: $D_{(j,j)} = D_{(j,j)} - (U_{(j,k)} \times V_{(j,k)}^T) \times (U_{(j,k)} \times V_{(j,k)}^T)^T$. The operation is a symmetric rank- k update.

HCORE_DGEMM: The operation applies updates to an off-diagonal low-rank tile of the input matrix, resulting from factorization of the low-rank tiles to the left of it. The operation involves two QR factorizations, one reduced SVD (with a rank truncation depending on the fixed rank and/or the fixed accuracy operation modes) and two matrix-matrix multiplications.

The most called computational kernel is HCORE_DGEMM and it also represents the one with highest arithmetic intensity. Once the sequential version of the code based on nested loops has been designed, we need to schedule the four aforementioned computational tasks on the underlying processing units.

8.2 Task-Based Programming Model

Task-based programming models have become methods of choice when targeting efficient parallel implementation, as they permit asynchronous thread executions after exposing fine-grained computational tasks. Static scheduling may be suboptimal here, especially in fixed accuracy mode, as this may result in load imbalance between tasks. Therefore, dynamic runtime systems are crucial to cope with the various tasks' workloads, besides handling dynamic frequency scaling of processors at runtime. Many dynamic runtime systems such as QUARK [26], StarPU [8], and OmpSs [12] exist for shared-memory systems. We use the task-based programming model and the dynamic runtime system, as implemented in OpenMP, for easy portability across hardware platforms. The TLR matrix generation and compression consist in generating the TLR matrix after performing an SVD using DGESVD on all off-diagonal tiles in an embarrassingly parallel

Algorithm 1. HiCMA_DPOTRF(HicmaLower, D, U, V, N, nb, rank, acc)

```

p = N / nb
for k = 1 to p do
  #pragma omp task depend(inout:D(k,k))
  hcore_dpoftrf(HicmaLower, D(k,k), rank, acc)
  for i = k+1 to p do
    #pragma omp task depend(in:D(k,k)) depend(inout:U(i,k))
    hcore_dtrsm(V(i,k), D(k,k), rank, acc)
  end for
  for j = k+1 to p do
    #pragma omp task depend(in:U(j,k)) depend(in:V(j,k)) depend(inout:D(j,j))
    hcore_dsyrk(D(j,j), U(j,k), V(j,k), rank, acc)
    for i = j+1 to p do
      #pragma omp task
      depend(in:U(i,k)) depend(in:V(i,k))
      depend(in:U(j,k)) depend(in:V(j,k))
      depend(inout:U(i,j)) depend(inout:V(i,j))
      hcore_dgemm(U(i,k), V(i,k), U(j,k), V(j,k), U(i,j), V(i,j), rank, acc)
    end for
  end for
end for

```

fashion using the parallel for loops from OpenMP. The QR-based DGESVD is slower than the divide-and-conquer DGESDD but requires much less memory. Other SVD variants (e.g., randomized SVD) may directly generate the TLR data format without going to the dense representation. These variants may also overcome these performance issues but this is beyond the scope of this paper. Algorithm 1 shows the pseudo-code of the TLR Cholesky factorization for the lower triangular case. Each kernel call is annotated by pragmas describing the data directions from which the compiler is capable of tracking the data dependencies. Each kernel's API has extra parameters related to fixed rank and/or fixed accuracy, allowing an algorithmic flexibility for end-users. The TLR compression and Cholesky factorization is currently being packaged into the Hierarchical Computations on Manycore Architectures (HiCMA) library and will be released during 2017.

9 Performance Results and Analysis

This section presents the performance results and analysis of the TLR compression and Cholesky factorization in the context of a climate/weather modeling application based on geospatial statistics.

9.1 Environment Systems

We have ported our OpenMP-based TLR compression and Cholesky factorization to three systems. We have considered three systems representative of

the current manycore-based hardware trends. The first system is composed of dual-socket 18-core Intel(R) Xeon(R) Haswell CPU E5-2699 v3 @ 2.3 GHz with 256 GB of main memory. The second system hosts the latest Intel commodity chip with dual-socket 14-core Intel(R) Xeon(R) Broadwell CPU E5-2680 v4 @ 2.4 GHz with 128 GB of main memory. The third system has the latest Intel(R) Xeon Phi(TM) Knights Landing manycore 7210 chips with 64 cores @ 1.30 GHz with 128 GB of main memory, operating in quadrant/cache modes. For simplicity, each system is named after its chip codename. Our TLR implementations have been compiled with Intel C compiler v16 and linked against sequential Intel MKL v11.3.1. We have run ten times each test configuration and report the average time as the consistent metric.

9.2 Memory Footprint Assessment

Theoretical Memory Footprint for Fixed Rank. For native dense Cholesky factorization, the memory footprint of the input matrix is simply $\frac{N^2}{2}$. For TLR Cholesky factorization, assuming fixed rank, the memory footprint can be calculated as follows. The numbers of diagonal and off-diagonal tiles are $ndt = \frac{N}{nb}$ and $nodt = \frac{(ndt * ndt) - ndt}{2}$, respectively. Therefore, assuming double precision and given a rank k , the required memory footprint for TLR is $8 * (ndt * \frac{nb * (nb + 1)}{2} + 2 * nodt * nb * k) \approx 4 * ndt * nb^2 + 16 * nodt * nb * k$.

Actual Memory Footprint for Fixed Accuracy. Figure 11(a) shows the memory footprint for dense and TLR Cholesky factorization up to $200K \times 200K$ covariance matrix size. The fixed accuracy of 10^{-9} is used, as required by the application. As seen in the figure, the TLR-based compression scheme exhibits more than an order of magnitude memory footprint saving with respect to naive dense Cholesky factorization.

Actual Operation Count for Fixed Accuracy. Figure 11(b) shows the operation count performed by dense and TLR Cholesky factorization up to $200K \times 200K$ covariance matrix size. Similarly, the fixed accuracy of 10^{-9} is used. As seen in the figure, the TLR Cholesky requires significantly less number of operations with respect to naive dense Cholesky factorization.

In both Figs. 11(a) and (b), the data points for matrix size of $73984 = 16^2 * 17^2$ are below the general trend for TLR-based scheme. This finding can be attributed to the better compression effect of the global Morton ordering for matrix sizes that are multiple of power of 2, as explained at the end of Sect. 4.

9.3 Performance of TLR Compression

Although the compression phase is important, it is performed only once, while generating the covariance matrix on the fly. Figure 12 reports the performance and scalability of TLR compression on the Haswell system for various tile sizes. We benchmark both DGESVD and DGESDD as MKL SVD implementations. DGESDD is faster thanks to its efficient divide-and-conquer at the expense of

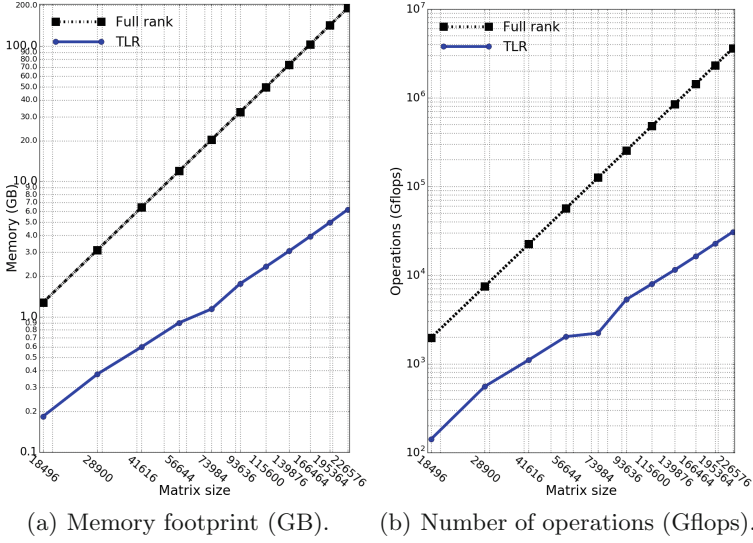


Fig. 11. Memory footprint and number of flops (accuracy is set to $1e-9$).

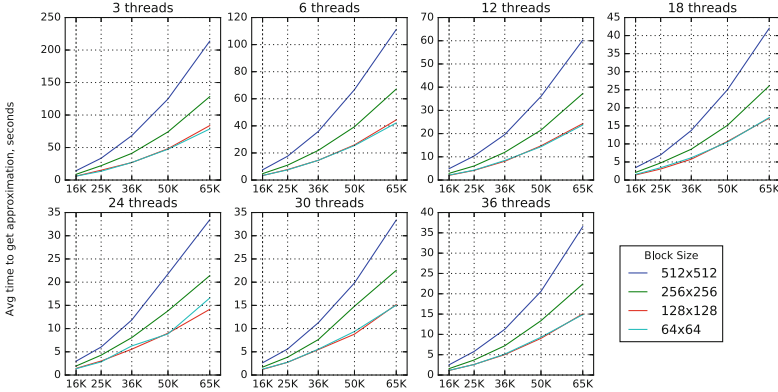


Fig. 12. Time to solution to approximate all tiles of a TLR matrix by DGESDD on various numbers of threads and block sizes.

requiring eight times more memory allocation than DGESVD. This explains the increase in time of approximation routine, when increasing the number of threads. TLR matrices with larger tiles tend to use more memory per tile and may saturate the memory bus bandwidth on the system due to the memory-bound character of the approximation phase. The scalability may be further improved through cross approximation techniques or randomized SVD kernel instead of ordinary dense SVD.

9.4 Performance of TLR Cholesky Factorization on Climate/Weather Modeling Applications

Figure 13 depicts the time to solution of the TLR Cholesky factorization (referred to as TLR-HiCMA_dpotrf) on various hardware architectures using an accuracy of 10^{-9} , as required by the application. In this figure, the time for compression has not been included, since this initial phase may only be done once before

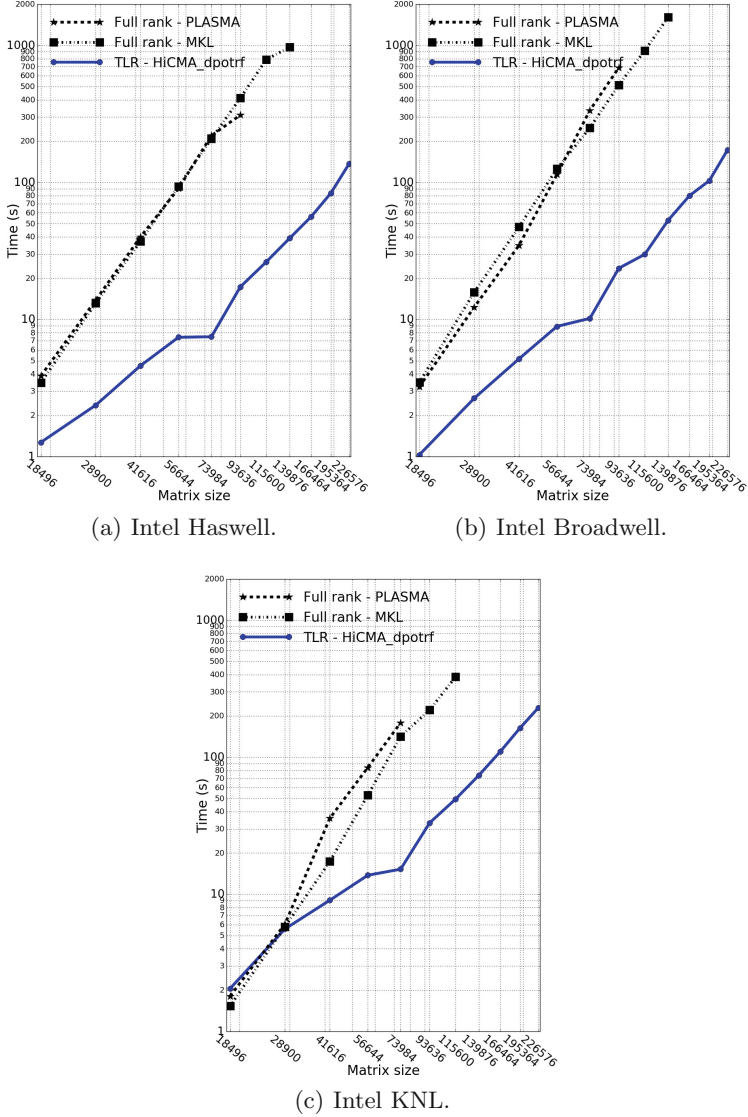


Fig. 13. Time to solution for TLR Cholesky factorization using an accuracy of 10^{-9} .

the matrix computation starts. Optimal tile sizes nb have been selected from empirical experiments for each Cholesky factorization variant, depending on the matrix size (e.g., for TLR Cholesky, $nb = 1156$ turns out to be the most effective). The naive interface of `PLASMA_dpotrf` call requires an out-of-place data translation, which doubles the memory footprint and prevents PLASMA from further scaling up. For all experiments in this section, we used almost the whole systems' resources as described in Sect. 9.1, except one or two cores, which are left to ensure that basic tasks of the operation system do not interfere with our experiments. As seen in Fig. 13, there is more than an order of magnitude time difference between TLR and dense Cholesky factorizations across all architectures. Some data points are missing for the dense approaches due to physical memory capacity. It is noteworthy to mention that the ranks after TLR compression have slightly grown after TLR Cholesky factorization, especially the off-diagonal tiles located at the bottom right. These tiles are the most manipulated tiles and receive updates throughout the TLR Cholesky factorization. Regarding the three architectures, the elapsed times of the full dense Cholesky factorization, as implemented in MKL and PLASMA on KNL, are considerably lower than those obtained for Haswell and Broadwell systems, thus showing the compute capability of KNL. However, the elapsed time of TLR Cholesky is slightly higher on KNL than those obtained on the other two architectures, due to the low arithmetic intensity of the sequential kernels. Moreover, we rely on the OpenMP dynamic runtime system to schedule the tasks on the different systems. While this shows decent performance on the commodity Intel CPU architectures with two sockets for which all cores share the same L3 cache, it does suffer from performance loss on KNL, for which only two cores share the same L2 cache. The overhead of moving data becomes a bottleneck, while performing work stealing across the higher core count KNL chip. A more regular static scheduling with data locality may perform better for such architecture. One can also notice that the time to solution of TLR Cholesky for the matrix size of $73984 = 17^2 16^2$ is relatively less with respect to the problem size growth. The reason is that the obtained rank after compression and total number of operations for this matrix size are less, as previously seen in Figs. 11(a) and (b).

10 Conclusion and Future Work

We have presented the tile low rank (TLR) Cholesky factorization in the context of climate/weather modeling application based on geospatial statistics on a Gaussian covariance matrix of size up to $200K \times 200K$. Our TLR Cholesky factorization achieves more than an order of magnitude in memory footprint saving and time to solution compared to native dense Cholesky factorization, as implemented in vendor optimized Intel MKL [19] and open-source PLASMA [27] libraries. Although TLR does not exhibit the best theoretical bounds for \mathcal{H} -matrix computations, it can still leverage, with a better user-productivity, a wide number of covariance-based applications toward much challenging hardware machines such as distributed-memory systems equipped with hardware

accelerators so that larger-scale problem dimensions can be covered. This will lead to new scientific research opportunities, especially for simulation workloads relying on the mainstream *R* software project. Moving forward, we would like to investigate batch algorithms by redesigning the current TLR Cholesky factorization from a tile-centric to a kernel-centric representation. This will help in compensating the kernel launch overhead due to the low arithmetic intensity, while increasing the hardware occupancy.

Acknowledgment. We would like to thank R. Kriemann from Max Planck Institute for Mathematics in the Sciences and M. Genton, A. Litvinenko, Y. Sun, and G. Turkiyyah from KAUST for fruitful discussions. We would like also to thank A. Heinecke from Intel for helping us tuning the codes on KNL. This work has been partially funded by the Intel Parallel Computing Center Award.

References

1. The R Project for Statistical Computing (2016). r-project.org
2. Agullo, E., Demmel, J., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., Ltaief, H., Luszczek, P., Tomov, S.: Numerical linear algebra on emerging architectures: the PLASMA and MAGMA projects. *J. Phys. Conf. Ser.* **180**, 012037 (2009)
3. Ambikasaran, S., Darve, E.: An $\mathcal{O}(N \log N)$ fast direct solver for partial hierarchically semiseparable matrices. *J. Sci. Comput.* **57**(3), 477–501 (2013)
4. Amestoy, P., Ashcraft, C., Boiteau, O., Buttari, A., L’Excellent, J.Y., Weisbecker, C.: Improving multifrontal methods by means of block low-rank representations. *SIAM J. Sci. Comput.* **37**(3), A1451–A1474 (2015)
5. Amestoy, P.R., Duff, I.S., L’Excellent, J.Y.: Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.* **184**(2), 501–520 (2000)
6. Aminfar, A., Darve, E.: A fast sparse solver for finite-element matrices. [arXiv:1403.5337](https://arxiv.org/abs/1403.5337) [cs.NA], pp. 1–25 (2014)
7. Anderson, E., Bai, Z., Bischof, C.H., Blackford, L.S., Demmel, J.W., Dongarra, J.J., Croz, J.J.D., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.C.: LAPACK User’s Guide, 3rd edn. SIAM, Philadelphia (1999)
8. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurr. Comput.: Pract. Exp.* **23**(2), 187–198 (2011)
9. Börm, S.: H2Lib 2.0. Max-Planck-Institut, Leipzig (1999–2012)
10. Börm, S.: Efficient numerical methods for non-local operators: \mathcal{H}^2 -Matrix compression, algorithms and analysis. EMS Tracts in Mathematics, vol. 14. European Mathematical Society, Zürich (2010)
11. Duputel, Z., Rivera, L., Fukahata, Y., Kanamori, H.: Uncertainty estimations for seismic source inversions. *Int. Geophys. J.* **190**(2), 1243–1256 (2012)
12. Duran, A., Ferrer, R., Ayguadé, E., Badia, R.M., Labarta, J.: A proposal to extend the OpenMP tasking model with dependent tasks. *Int. J. Parallel Prog.* **37**(3), 292–305 (2009)
13. The FLAME project, April 2010. <http://z.cs.utexas.edu/wiki/flame/wiki/FrontPage>
14. Hackbusch, W.: A sparse matrix arithmetic based on \mathcal{H} -matrices. Part i: introduction to \mathcal{H} -matrices. *Computing* **62**(2), 89–108 (1999)

15. Hackbusch, W., Börm, S.: Data-sparse approximation by adaptive \mathcal{H}^2 -matrices. *Computing* **69**(1), 1–35 (2002)
16. Hackbusch, W., Khoromskij, B., Sauter, S.: On \mathcal{H}^2 -Matrices. In: Bungartz, H.J., Hoppe, R., Zenger, C. (eds.) *Lectures on Applied Mathematics*, pp. 9–29. Springer, Heidelberg (2000)
17. Hackbusch, W.: *Hierarchical Matrices: Algorithms and Analysis*, vol. 49. Springer, Heidelberg (2015)
18. Hackbusch, W., Börm, S., Grasedyck, L.: *HLib 1.4*. Max-Planck-Institut, Leipzig (1999–2012)
19. Intel: Math Kernel Library (2016). software.intel.com/en-us/intel-mkl
20. Kriemann, R.: \mathcal{H} -LU factorization on many-core systems. *Comput. Vis. Sci.* **16**(3), 105–117 (2013)
21. Ltaief, H., Gratadour, D., Charara, A., Gendron, E.: Adaptive optics simulation for the world’s largest telescope on multicore architectures with multiple GPUs. In: *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2016*. pp. 9:1–9:12. ACM, New York (2016)
22. Meuer, H., Strohmaier, E., Dongarra, J., Simon, H.: The Top500 List, November 2016. <http://www.top500.org>
23. Rouet, F.H., Li, X.S., Ghysels, P., Napov, A.: A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. *ACM Trans. Math. Softw.* **42**(4), 27:1–27:35 (2016)
24. Sun, Y., Stein, M.L.: Statistically and computationally efficient estimating equations for large spatial datasets. *J. Comput. Graph. Stat.* **25**(1), 187–208 (2016)
25. Tyrtyshnikov, E.E.: Mosaic-skeleton approximations. *Calcolo* **33**(1), 47–57 (1996)
26. YarKhan, A., Kurzak, J., Dongarra, J.: QUARK users’ guide: Queueing and runtime for kernels. Technical report ICL-UT-11-02, University of Tennessee Innovative Computing Laboratory (2011)
27. YarKhan, A., Kurzak, J., Luszczek, P., Dongarra, J.: Porting the PLASMA numerical library to the OpenMP standard. *Int. J. Parallel Program.* **45**(3), 612–633 (2017). doi:[10.1007/s10766-016-0441-6](https://doi.org/10.1007/s10766-016-0441-6)

High Performance Computing
32nd International Conference, ISC High Performance
2017, Frankfurt, Germany, June 18–22, 2017,
Proceedings
Kunkel, J.; Yokota, R.; Balaji, P.; Keyes, D. (Eds.)
2017, XV, 432 p. 174 illus., Softcover
ISBN: 978-3-319-58666-3