Flexible Data Redistribution in a Task-Based Runtime System

Qinglei Cao[1,*], George Bosilca[1], Wei Wu[2], Dong Zhong[1], Aurelien Bouteiller[1], and Jack Dongarra[1,3]

[1]University of Tennessee, US
[2]Los Alamos National Laboratory, US
[3]University of Manchester, UK
[*]*qcao3@vols.utk.edu*

***Abstract*—Data redistribution aims to reshuffle data to optimize some objective for an algorithm. The objective can be multi-dimensional, such as improving computational load balance or decreasing communication volume or cost, with the ultimate goal to increase the efficiency and therefore decrease the time-to-solution for the algorithm. The classical redistribution problem focuses on optimally scheduling communications when reshuffling data between two regular, usually block-cyclic, data distributions. Recently, task-based runtime systems have gained popularity as a potential candidate to address the programming complexity on the way to exascale. In addition to an increase in portability against complex hardware and software systems, task-based runtime systems have the potential to be able to more easily cope with less-regular data distribution, providing a more balanced computational load during the lifetime of the execution.**

In this scenario, it becomes paramount to develop a general redistribution algorithm for task-based runtime systems, which could support all types of regular and irregular data distributions. In this paper, we detail a flexible redistribution algorithm, capable of dealing with redistribution problems without constraints of data distribution and data size and implement it in a task-based runtime system, `PaRSEC`. Performance results show great capability compared to `ScaLAPACK`, and applications highlight an increased efficiency with little overhead in terms of data distribution and data size.

***Index Terms*—Data redistribution, Data size, Task-based runtime system, High-performance computing**

I. Introduction

In many scientific applications, data needs to be frequently moved from one distribution scheme into another at runtime, in order to provide better data locality, load balance, as well as performance. For instance, in adaptive mesh refinement (`AMR`), in order to dynamically adapt the accuracy of a solution within certain sensitive or turbulent regions of simulation, these regions need to be refined; hence redistribution is always applied on these regions with the explicit goal of a better load balance. Actually, the question of data redistribution has been proposed for more than two decades, both statically and dynamically, as this question was central when dealing with the imposed data distributions of early distributed-memory programming models such as High Performance Fortran (`HPF`) [1], and has received significant attention. Array redistribution, popular in `HPF`, used to dynamically change the distribution of an array from a specified source distribution to a specified target distribution, is one of the most expensive communication patterns, and is particularly important for applications where the parallelism alternates between dimensions of the data. As a result, numerous scientific literature on array redistribution exists [2]–[7]. More general data redistribution focuses on redistribution between two data sets (e.g., from how it was generated by the producer to how the application needs the data to be laid out among its processes [8]) or relocating data distributed across one producer grid onto a different distribution scheme across a consumer grid [9].

Research on redistribution involves not only `HPF` but also the Message Passing Interface (`MPI`) [6], [10], towards both coarse-grained [11] and fine-grained [12], [13] for many scientific domains—including linear algebra, like `ScaLAPACK` [14], [15], and particle codes [12], [13], [16]. However

- these approaches usually focus on regular data distributions—the static two-dimensional block cyclic data distribution (2DBCDD) descriptor on which the dense linear algebra community has been relying for more than two decades. Irregular data distribution is also important from a load balancing perspective in terms of memory, computation and communication, as suggested by the hybrid data distribution (called "band distribution") utilized in [17] used for tiled low-rank (`TLR`) Cholesky.
- distribution is the ultimate goal for these studies (even if derived data size changes as side-effect like in `ScaLAPACK` [18]); in fact, besides distribution, finding the right data size (a.k.a. tile size in tile-based algorithm like `PLASMA` [19] and `DPLASMA` [20])—the one that trades-off performance and level of concurrency—is also a critical step [21]. In many cases, e.g. [22], the so-called data tiling size is critical and dependent on the problem size, and has been elusive to determine *a single best data size* used for the whole linear algebra system including multiple stages.

Due to the increasing complexity of hardware architectures and communication topologies, many of the regular data distributions might be unfitting for modern problems, both in terms of the efficiency and the scalability of the resulting algorithms. Moreover, as the popularity of task-based runtimes increases, it is interesting to revisit the data distribution problem in their context, and imagine support for more flexible, possibly less regular, data distributions in a task-based runtime system. In this paper, we propose a flexible redistribution algorithm which could solve a general data redistribution problem and evaluate this algorithm in a task-based runtime system, i.e., `PaRSEC`. To our knowledge, this is the first time a general redistribution

978-1-7281-6677-3/20/$31.00 ©2020 IEEE
DOI 10.1109/CLUSTER49012.2020.00032

TABLE I: Parameters and notations.

Symbol	Description
$\mathcal{R}$	Function or routine for redistribution
SRC	Source data descriptor
TG	Target data descriptor
A_{sub}	Submatrix to be redistributed
$size_{\{row,col\}}$	Row/column size of A_{sub}
$disp\_row_{\{s,t\}}$	Row displacement in source(s)/target(t)
$disp\_col_{\{s,t\}}$	Column displacement in source(s)/target(t)
$\{M,N\}_{\{s,t\}}$	Row(M)/column(N) size of source(s)/target(t)
$\{MB,NB\}_{\{s,t\}}$	Row(MB)/column(NB) tile size of source(s)/target(t)
$D_{\{s,t\}}$	Data distribution of source(s)/target(t)
$\{m,n\}_{\{s,t\}}$	Tile row(m)/column(n) index of source(s)/target(t)
local	Source and target data on the same process
remote	Source and target data on different processes
SEGMENTS	**NW**, **N**, **NE**, **W**, **I**, **E**, **SW**, **S**, and **SE**

algorithm has been proposed in task-based runtime worlds.

The remainder of this paper is as follows. Section II introduces the design and the implementation of the redistribution algorithm in PaRSEC. Performance results and analysis, along with application demonstrations, are illustrated in Section III. Section IV presents related work, and we conclude and present future work in Section V.

II. Redistribution

A. Problem Definition

A general redistribution problem $\mathcal{R}$ is a function or routine to change distribution schemes (Table I describes parameters and notations): $\mathcal{R}$: SRC $\rightarrow$ TG with the following properties:

- Source SRC with the distribution D_s, and target TG with the distribution D_t;
- Submatrix A_{sub} to be redistributed with size of $size\_row \times size\_col$ and with displacements ($disp\_row_s$, $disp\_col_s$) in SRC and ($disp\_row_t$, $disp\_col_t$) in TG, and A_{sub} should not exceed the bounds of SRC and TG.

Figure 1 depicts a general redistribution problem, redistributing a submatrix from SRC to TG with different distributions and tile sizes. While the problem is generic, in our particular context A_{sub} is to be redistributed between two matrices stored in tile format, using the data descriptor in PaRSEC. There are several features that need to be clarified:

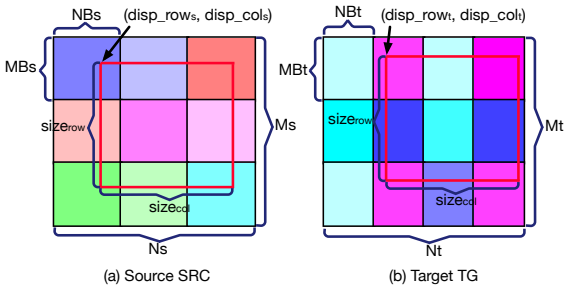


Fig. 1: General redistribution problem; matrix is stored in tile format, each color represents a different process, and rectangle circled in red is the submatrix to be redistributed.

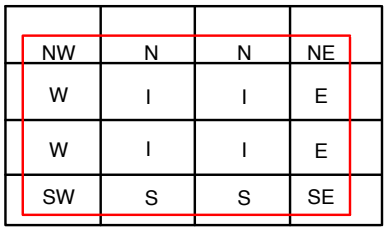


Fig. 2: The red rectangle represents one TG tile while black rectangles are the corresponding SRC tiles.

- Tiles in SRC and TG are rectangles, not specified as square, and MB_s, NB_s, MB_t, NB_t are independent with D_s and D_t;
- Displacements of ($disp\_row_s$, $disp\_col_s$) in SRC and ($disp\_row_t$, $disp\_col_t$) in TG could be any points not exceeding the bounds of SRC or TG respectively.

B. Algorithm Description

For a general redistribution problem, to redistribute a submatrix between two matrices with different distributions, tile sizes, and displacements, an efficient algorithm should be as flexible as possible to deal with all possible cases. Hence, as shown in Figure 2, zooming in one tile in TG to catch its source data in SRC, we split the TG tile into 9 parts, or SEGMENTS, according to their location in SRC, **N**orth**W**est (**NW**), **N**orth (**N**), **N**orth**E**ast (**NE**), **W**est (**W**), **I**nner (**I**), **E**ast (**E**), **S**outh**W**est (**SW**), **S**outh (**S**) and **S**outh**E**ast (**SW**). Figure 3 shows the possible categories based on the existence of SEGMENTS, determined by combinations of $size\_row$, $size\_col$, MB_s, NB_s, MB_t, NB_t and location of TG tiles' starting points in SRC. All possible cases of general redistribution problems are extensions of these 9 categories, including several **N**, **S**, **W**, **E** or **I**, e.g. Figure 2 is an extension of Figure 3 (8).

The serial algorithm, revealed in Algorithm 1, follows the idea that for tiles in TG, send/receive (when *remote*) or copy (when *local*) SEGMENTS. The benefits of this design are that it: (1) is capable for coarse- and fine- grained redistribution problems for tile- or block- based matrix partition; (2) isolates distribution and tile size. Actually, it could solve a redistribution problem with absolute flexibility on distribution, tile size and

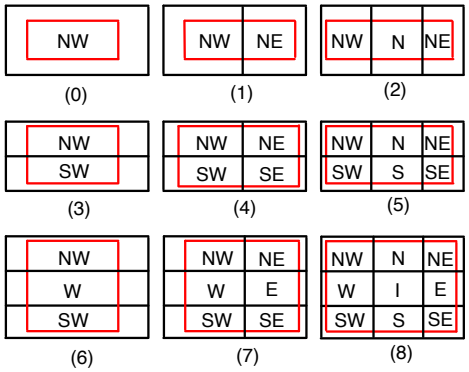


Fig. 3: All possible 9 categories; red rectangles represent one TG tile while black rectangles are the corresponding SRC tile(s).

Algorithm 1 Serial Algorithm of Redistribution

```
for m_t = disp_row_t/MB_t to (size_row +disp_row_t-1)/MB_t do
  for n_t = disp_col_t/NB_t to (size_col +disp_col_t-1)/NB_t do
    Calculate m_s_start, m_s_end, n_s_start, and n_s_end that (m_t,
    n_t) associated with
    for m_s = m_s_start to m_s_end do
      for n_s = n_s_start to n_s_end do
        if Remote then
          Send SEGMENTS
        end if
      end for
    end for
    if Remote then
      Receive SEGMENTS
    else
      Copy SEGMENTS
    end if
  end for
end for
```

displacement. In this way, all possible redistribution problems could be reduced to a combination of these 9 SEGMENTS, and operations on these SEGMENTS could be considered as tasks which thus could be efficiently handled by a multi-threaded task-based runtime system.

C. The PaRSEC Runtime System

PaRSEC [23] is a generic task-based runtime system for asynchronous, architecture-aware scheduling of fine-grained tasks on distributed many-core heterogeneous architectures. It is capable of dynamically unfolding a concise description of a graph of tasks on a set of resources and satisfying all data dependencies by efficiently shepherding data between memory spaces (between nodes but also between different memories on different devices) and scheduling tasks across heterogeneous resources. Several domain-specific languages (DSLs) [24] in PaRSEC, such as Parameterized Task Graph (PTG) [25] and Dynamic Task Discovery (DTD) [26], help domain scientists to focus only on their domain knowledge instead of low-level computer science aspects, such as the complex hardware architectures, hierarchical memory layout, different types of communication prototypes, etc.

D. Implementation in PaRSEC Runtime System

To implement the Algorithm 1 in PaRSEC and expose all potential parallelism, four different types of tasks (a.k.a task classes) are specified:

- Init: prepare TG data for tasks in task classes Receive and Finish to protect it from being simultaneously modified by multiple tasks within a process;
- Send: send data if *remote* and pass the address if *local*;
- Receive: receive data if *remote* or copy data if *local* to the target data descriptor TG;
- Finish acts as synchronization for each tile in TG to finish all related tasks in Receive.

For the purpose of data locality, Send is local to SRC's tiles, while Init, Receive, and Finish reside on TG's tiles. Several runtime-level optimizations are proposed to efficiently utilize network bandwidth, so that to get higher performance, but in this paper we only focus on the algorithm itself.

III. Performance Results and Analysis

A. Experiments Settings

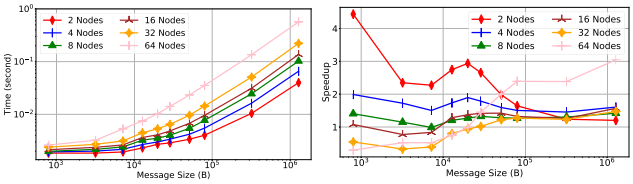
Experiments are conducted on two HPC clusters: NaCL and Shaheen II, and assumes no memory constrain for data redistribution. NaCL includes 66 compute nodes connected by InfiniBand QDR, and each node has two 2.8 GHz Intel Xeon X5660. Shaheen II is a Cray XC40 system with 6,174 compute nodes; each node is equipped with two 16-core Intel Haswell CPUs running at 2.30 GHz and 128 GB DDR4 RAM; the interconnect is Cray Aries with Dragonfly topology.

B. Comparison to ScaLAPACK

ScaLAPACK is a high-performance library for linear algebra routines. ScaLAPACK's data format is inherited from LAPACK [27], but it's targeted to parallel distributed memory machines instead. We compare our implementation in PaRSEC to redistribution routines in ScaLAPACK. It should be noted that ScaLAPACK only support redistribution between regular, block-cyclic, data distributions, so we restrict the scope of this evaluation to such data distributions. Figure 4 shows a weak-scaling experiment, same matrix size per node for all number of nodes; (a) presents ScaLAPACK execution time for the redistribution process and (b) the speedup of our implementation compared to ScaLAPACK. ScaLAPACK behaves better for small message size, especially with a larger number of nodes. Because runtime overheads exist in task-based runtime systems like PaRSEC but not in ScaLAPACK, which becomes increasingly dominant in a weak scaling experiment, as the actual execution time is very small as shown in Figure 4 (a). In fact, small task granularity is not the most suitable setup for a task-based runtime systems [28]. As the message size grows, the speedup is almost constant on different number of nodes (there is unknown issue for ScaLAPACK on 64 nodes).

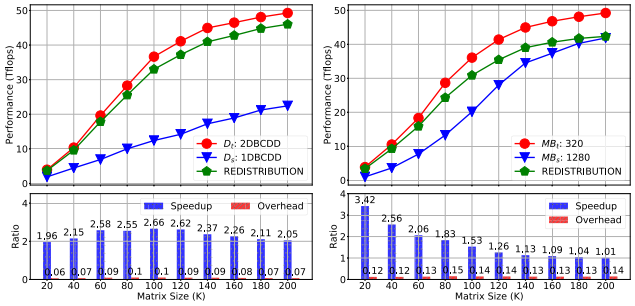
C. Benefits and Overheads in Real Applications

Cholesky factorizations is a widely used algorithms to solve linear systems of equations ($Ax = B$). We use tiled dense Cholesky from DPLASMA and TLR Cholesky from Lorapo [17], both using the PaRSEC runtime system, to evaluate benefits and overheads of redistribution. We evaluate a case where the data generator provides the data in a distribution that is inappropriate and would result in an inefficient execution, and where a redistribution of the data could result in a more efficient execution. The following figures present effects of redistribution on two different setups, converting data distribution and tile size. These optimization may be combined in practice [17], [22].



(a) Execution time of ScaLAPACK (b) Speedup to ScaLAPACK

Fig. 4: Comparison to ScaLAPACK on NaCL.

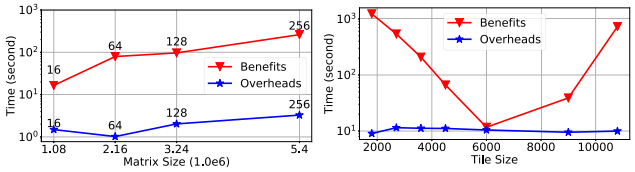


(a) Effect of data distribution. (b) Effect of tile size conversion.

Fig. 5: Cholesky factorization on `Shaheen II` (64 nodes).

Figure 5 (a) showcases a data redistribution maintaining the tile size ($MB_s = MB_t = 320$). The source matrix, a 1DBCDD with D_s: $(P, Q) = (1, 64)$, does not expose enough parallelism, and thus exhibits poor performance. A much more suitable distribution for this case, known theoretically but also highlighted in the Figure, would be 2DBCDD with D_t: $(P, Q) = (8, 8)$. The "REDISTRIBUTION" redistributes from D_s to D_t, executes the Cholesky factorization, and then redistributes the matrix back to D_s, such that the entire redistribution is transparent to the caller. In Figures 5 (b), the data distribution is fixed to 2DBCDD but $MB_s = 1280$ is suboptimal when matrix is small as it reduces the parallelism, and hinders performance. Similar to above, the "REDISTRIBUTION" redistributes the matrix from MB_s to $MB_t = 320$, executes the Cholesky factorization, and redistributes matrix back to MB_s. From Figure 5 (a) and (b), the "REDISTRIBUTION" can automatically convert the matrix into a more suitable data distribution 2DBCDD and/or tile size, with little overheads (less than 14%) allowing the execution to unfold to be most favorable setup on the platform.

Figure 6 depicts a similar experiment using `TLR` Cholesky factorization where both distribution [17] and tile size [22] are critical. (a) shows the impact of the data distribution while maintaining the tile size, where the D_s is 2DBCDD and D_t is less regular, a 2DBCDD distribution with a band of tiles around the diagonal in a 1DBCDD distribution ("band distribution", the benefits for such a distribution are analyzed in [22]). A kind of modified "weak scaling" in terms of memory constrain is deployed on 16, 64, 128 and 256 nodes for **st-3D-sqexp** [17]. (b) depicts the impact of tile size changes (MB_s varies while $MB_t = 5400$) while maintaining a similar data distribution for a matrix of $2.16M \times 2.16M$ elements on 16 nodes for **syn-2D** [17]. From these two figures, overheads (execution time of only calling data redistribution) are small compared to the benefits (execution time of "$D_s - D_t$" in Figure 6 (a) and "$MB_s - MB_t$" in Figure 6 (b)).



(a) Effect of data distribution. (b) Effect of tile size conversion.

Fig. 6: `TLR` Cholesky on `Shaheen II`.

These figures show domain scientists do not have anymore to stick with predefined data distributions, that impact the data generation potential, but instead, for a reasonable overhead, allow a mismatch between data generators and users to happen.

IV. Related Work

For more than three decades, research on data redistribution has evolved around regular data distributions. In the 1990s, research about array and data redistribution sprung up after the appearance of `HPF` [2], [4], [5], [29]. In the 2000s, research spread to more broad fields [12], [30], [31]. More recently, we witnessed a resurgence of interest in data redistribution due to increasingly complex applications which need to improve data locality and/or reduce cost of data movement therefore to relocate data distributed across one grid onto a different distribution scheme across another grid [8], [9], [11], [13], [32]. However, all these researches on array or data redistribution: (1) focused on a simplified problem, aka regular 2DBCDD distribution; (2) tried to address load imbalance caused by the data distribution, but ignored impact from data size. They also highlighted that with the increasing complexity of hardware architectures and communication topologies, targeting only on regular 2DBCDD distribution is not enough. As task-based runtime systems emerge, a general redistribution algorithm, taking in account not only regular and irregular data distribution but also the impact of data size, becomes necessary.

V. Conclusion and Future Work

This paper presents a flexible and general redistribution algorithm for task-based runtime system, supporting any regular and irregular data distributions. We provide an implementation in a task-based runtime `PaRSEC`, and the practical evaluation of our implementation shows it can achieve better performance compared with existing tools supporting some level of data redistribution, `ScaLAPACK`. Moreover, utilization in real applications highlights great benefits and negligible overheads in terms of data distribution and tile size with significant improvement in application time-to-solution.

For future work, we plan to explore the applicability of this redistribution algorithm to other runtime systems. In the context of `PaRSEC`, we plan to further reduce communication overheads and make the redistribution a completely transparent process, an operation that could be fused either with the ensuing computation, to hide all overheads related to the redistribution in terms of memory and time-to-solution.

VI. Acknowledgments

This research was supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. The authors would like also to thank Cray Inc. and Intel in the context of the Cray Center of Excellence and Intel Parallel Computing Center awarded to the Extreme Computing Research Center at KAUST. For computer time, this research used the *Shaheen-2* supercomputer hosted at the Supercomputing Laboratory at KAUST.

References

[1] D. B. Loveman, "High performance fortran," *IEEE Parallel & Distributed Technology: Systems & Applications*, vol. 1, no. 1, pp. 25–42, 1993.

[2] A. Wakatani and M. Wolfe, "A new approach to array redistribution: Strip mining redistribution," in *International Conference on Parallel Architectures and Languages Europe*. Springer, 1994, pp. 323–335.

[3] ——, "Optimization of array redistribution for distributed memory multicomputers," *Parallel Computing*, vol. 21, no. 9, pp. 1485–1490, 1995.

[4] S. Kaushik, C.-H. Huang, J. Ramanujam, and P. Sadayappan, "Multi-phase array redistribution: modeling and evaluation," in *Proceedings of 9th International Parallel Processing Symposium*. IEEE, 1995, pp. 441–445.

[5] R. Thakur, A. Choudhary, and J. Ramanujam, "Efficient algorithms for array redistribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 6, pp. 587–594, 1996.

[6] Y. W. Lim, P. B. Bhat, and V. K. Prasanna, "Efficient algorithms for block-cyclic redistribution of arrays," *Algorithmica*, vol. 24, no. 3-4, pp. 298–330, 1999.

[7] C.-H. Hsu, S.-W. Bai, Y.-C. Chung, and C.-S. Yang, "A generalized basic-cycle calculation method for efficient array redistribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 12, pp. 1201–1216, 2000.

[8] T. Marrinan, J. A. Insley, S. Rizzi, F. Tessier, and M. E. Papka, "Automated dynamic data redistribution," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* . IEEE, 2017, pp. 1208–1215.

[9] C. Foyer, A. Tate, and S. McIntosh-Smith, "Aspen: An efficient algorithm for data redistribution between producer and consumer grids," in *European Conference on Parallel Processing*. Springer, 2018, pp. 171–182.

[10] D. W. Walker and S. W. Otto, "Redistribution of block-cyclic data distributions using mpi," *Concurrency: Practice and Experience*, vol. 8, no. 9, pp. 707–728, 1996.

[11] A. Reisner, L. N. Olson, and J. D. Moulton, "Scaling structured multigrid to 500k+ cores through coarse-grid redistribution," *SIAM Journal on Scientific Computing*, vol. 40, no. 4, pp. C581–C604, 2018.

[12] M. Hofmann and G. Rünger, "Fine-grained data distribution operations for particle codes," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2009, pp. 54–63.

[13] ——, "Flexible all-to-all data redistribution methods for grid-based particle codes," *Concurrency and Computation: Practice and Experience* , vol. 30, no. 13, p. e4421, 2018.

[14] J. Dongarra, L. Prylli, C. Randriamaro, and B. Tourancheau, "Array redistribution in scalapack using pvm," in *EuroPVM*, vol. 95. Citeseer, 1995, pp. 271–276.

[15] L. Prylli and B. Tourancheau, "Efficient block cyclic data redistribution," in *European Conference on Parallel Processing*. Springer, 1996, pp. 155–164.

[16] M. Hofmann and G. Rünger, "Efficient data redistribution methods for coupled parallel particle codes," in *2013 42nd International Conference on Parallel Processing*. IEEE, 2013, pp. 40–49.

[17] Q. Cao, Y. Pei, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, "Extreme-scale task-based cholesky factorization toward climate and weather prediction applications," in *Proceedings of the Platform for Advanced Scientific Computing Conference* , 2020, pp. 1–11.

[18] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley, *ScaLAPACK Users' Guide*. Philadelphia: Society for Industrial and Applied Mathematics, 1997.

[19] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, "A class of parallel tiled linear algebra algorithms for multicore architectures," *Parallel Computing*, vol. 35, no. 1, pp. 38–53, 2009.

[20] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, A. Haidar, T. Hérault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. YarKhan, and J. Dongarra, "Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA," in *IPDPS Workshops*. IEEE, 2011, pp. 1432–1441. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6008655

[21] J. Kurzak, H. Anzt, M. Gates, and J. Dongarra, "Implementation and Tuning of Batched Cholesky Factorization and Solve for NVIDIA GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 2036–2048, July 2016.

[22] Q. Cao, Y. Pei, T. Herault, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. E. Keyes, and J. Dongarra, "Performance analysis of tile low-rank cholesky factorization using parsec instrumentation tools," in *SC'2019 Workshop on Programming and Performance Visualization Tools*, 2019.

[23] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J. Dongarra, "PaRSEC: A Programming Paradigm Exploiting Heterogeneity for Enhancing Scalability," *Computing in Science and Engineering*, vol. 99, p. 1, 2013.

[24] ——, "PaRSEC: Exploiting heterogeneity to enhance scalability," *Computing in Science Engineering*, vol. 15, no. 6, pp. 36–45, Nov 2013.

[25] A. Danalis, G. Bosilca, A. Bouteiller, T. Herault, and J. Dongarra, "PTG: An Abstraction for Unhindered Parallelism," 2014, pp. 21–30.

[26] R. Hoque, T. Herault, G. Bosilca, and J. Dongarra, "Dynamic Task Discovery in PaRSEC: A Data-flow Task-based Runtime," in *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ser. ScalA '17, 2017, pp. 6:1–6:8.

[27] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' guide*. Siam, 1999, vol. 9.

[28] E. Slaughter, W. Wu, Y. Fu, L. Brandenburg, N. Garcia, W. Kautz, E. Marx, K. S. Morris, W. Lee, Q. Cao, G. Bosilca, S. Mirchandaney, S. Treichler, P. S. McCormick, and A. Aiken, "Task bench: A parameterized benchmark for evaluating parallel runtime performance," *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'2020)*, 2020. [Online]. Available: http://arxiv.org/abs/1908.05790

[29] A. P. Petitet and J. J. Dongarra, "Algorithmic redistribution methods for block-cyclic decompositions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 12, pp. 1201–1216, 1999.

[30] M. Guo and I. Nakata, "A framework for efficient data redistribution on distributed memory multicomputers," *The Journal of Supercomputing*, vol. 20, no. 3, pp. 243–265, 2001.

[31] R. Sudarsan and C. J. Ribbens, "Efficient multidimensional data redistribution for resizable parallel computations," in *International Symposium on Parallel and Distributed Processing and Applications*. Springer, 2007, pp. 182–194.

[32] J. Herrmann, G. Bosilca, T. Hérault, L. Marchal, Y. Robert, and J. Dongarra, "Assessing the cost of redistribution followed by a computational kernel: complexity and performance results," *Parallel Computing*.