Performance Impact of Rank-Reordering on Advanced Polar Decomposition Algorithms

Aniello Esposito
Cray EMEA Research Lab (CERL)
Cray Computer GmbH
Basel, Switzerland
Email: esposito@cray.com

David Keyes, Hatem Ltaief, Dalal Sukkari
Extreme Computing Research Center
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
Email: david.keyes@kaust.edu.sa, hatem.ltaief@kaust.edu.sa
dalal.sukkari@kaust.edu.sa

***Abstract*—We demonstrate the importance of both MPI rank reordering and choice of processor grid topology in the context of advanced dense linear algebra (DLA) applications for distributed-memory systems. In particular, we focus on the advanced polar decomposition (PD) algorithm, based on the QR-based Dynamically Weighted Halley method (QDWH). The QDWH algorithm may be used as the first computational step toward solving symmetric eigenvalue problems and the singular value decomposition. Sukkari et al. (ACM TOMS, 2017) have shown that QDWH may benefit from rectangular instead of square processor grid topologies, which directly impact the performance of the underlying `ScaLAPACK` algorithms. In this work, we experiment an extensive combination of grid topologies and rank reorderings for different matrix sizes and number of nodes, and use QDWH as a proxy for advanced compute-bound linear algebra operations, since it is rich in dense linear solvers and factorizations. A performance improvement of up to 54% can be observed for QDWH on 800 nodes of a Cray XC system, thanks to an optimal combination, especially in strong scaling mode of operation, for which communication overheads may become dominant. We perform a thorough application profiling to analyze the impact of reordering and grid topologies on the various linear algebra components of the QDWH algorithm. It turns out that point-to-point communications may be considerably reduced thanks to a judicious choice of grid topology, while properly setting the rank reordering using the features from the `cray-mpich` library.**

***Keywords*-Performance Analysis; Rank Reordering; Grid Topology; Polar Decomposition; Strong Scaling;**

I. Introduction

A considerable amount of parallel applications in HPC employ point-to-point communication to exchange information between a pair of specific processing elements (PE), besides collective communication involving all the participants in a simulation. In cases where the point-to-point communication prevails, substantial slowdown and load imbalance may manifest, when communication paths unnecessarily cross compute node boundaries, due to limiting factors such as the network latency encountered for small messages and the injection bandwidth in general. An obvious improvement approach consists of maximizing the on-node communication, and therefore reduce the related off-node traffic, through appropriate placement of PEs on compute nodes. A common pattern for point-to-point communications consists in nearest neighbor interactions on a Cartesian grid topology, for instance, as seen for the iterative solution of partial differential equations or sparse matrix problems. While these applications are the typical beneficiaries of rank reordering in the strong scaling limit, it is less obvious that traditional dense linear algebra (DLA) applications can benefit as well, because of the surface-to-volume effect, i.e., performing O(n^3) operations on O(n^2) data, for which computation time is automatically assumed to be prevalent over communication and memory traffic. The `LINPACK` benchmark [1], which measures the sustained peak performance of a system for the Top500 list, is one of the most prominent examples. However, applications composed of successive calls to high-level DLA matrix operations of irregular workloads may also generate substantial network traffic and, thus, may suffer from process misplacement, especially in strong scaling mode of operations. In addition, load-balancing issues due to specific choices of grid topologies may appear and become cumbersome. Therefore, it is critical to mitigate the communication overheads by employing an optimal rank reordering with an appropriate PE placement.

In this work, we propose to perform a comprehensive performance analysis using the polar decomposition (PD) based on the advanced iterative QR-based Dynamically Weighted Halley (QDWH) algorithm, which successively calls dense matrix operations. The QDWH approach for the PD algorithm is also suited for symmetric eigensolvers and singular value decompositions, thanks to its high compute-intensiveness and degree of parallelism. However, in situations where PEs run out of work due to strong scaling, QDWH suffers from expensive communication overheads and load imbalance. Compared to our previous work [9], we thoroughly investigate the combined performance impact of rank reordering and processor grid topology on QDWH. In particular, we highlight the limit of the surface-to-volume effect and present a rank reordering strategy to overcome the performance bottlenecks. Since the existing QDWH

relies on ScaLAPACK [2], the linear algebra algorithms are decoupled from the actual data distribution, i.e., the two-dimensional block cyclic data distribution. This data distribution may be mapped to a two-dimensional Cartesian grid topology, with a number of row and column processors. Performance analysis reveals that point-to-point communications are one of the main limiting factors, especially when work is not sufficiently available to hide the data motion overheads. Using a rank reordering strategy from cray-mpich, combined with a judicious choice of grid topologies, the standard QDWH may benefit up to 54% of performance improvement on 800 nodes of a two-socket 16-core Intel Broadwell Cray XC system.

The remainder of the paper is as follows. Section II gives an overview of rank reordering capabilities of the cray-mpich on Cray systems. Section III recalls the description of the QDWH algorithm and highlights its various linear algebra operations used as building blocks. The simulation setup and results are then illustrated in Section IV. Section V provides a comprehensive application profiling analysis. Details on the software integration of QDWH in cray-libsci are given in Section VI. Section VII summarizes and presents future works.

II. Rank Reordering

An allocation of N_N compute nodes on a Cray system featuring cray-mpich is assumed, with an application launcher such as aprun or srun. Every compute node has at least N_C cores available and the application to be executed makes use of the message passing interface (MPI), where a PE is represented by an MPI rank. The number of cores per nodes as well as the total number of MPI ranks $N_P = N_N \times N_C$, can be factorized by two integers each, i.e. $N_P = P \times Q$ and $N_C = m_c \times n_c$, where m_c and n_c is a divisor of P and Q, respectively. If not all cores per node are used, aprun and srun distribute the MPI ranks differently by default but with the appropriate set of options an equivalent placement can be achieved. For both launchers, successive MPI ranks are placed on the same node by default which is referred to as the SMP-style placement of processing elements. The cray-mpich library allows to override the default MPI rank placement scheme by means of the MPICH_RANK_REORDER_METHOD environment variable. This runtime parameter accepts values from $0-4$, where 1 is the SMP scheme and 3 allows to specify a custom rank placement which is the focus of this work. Option 0 specifies a round-robin placement, where sequential MPI ranks are placed on the next node in the list. Option 2 specifies a folded-rank placement, where again sequential MPI ranks are placed on the next node in the list but when every node has been used, instead of starting over with the first node again, the rank placement starts at the last node, going back to the first. Finally, option 4 specifies a topology-aware rank placement. This option determines an optimized rank

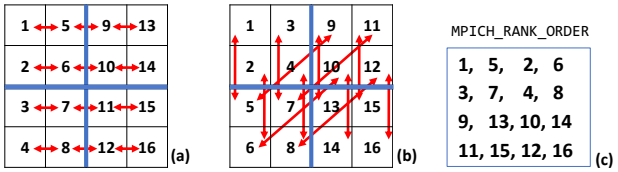


Figure 1: (a) Column-major global rank ordering used internally by an application. Blue lines denote the compute node boundaries and red arrows only the vertical communication for simplicity (color online). (b) SMP-style rank ordering used by default on the Cray XC system. Red arrows correspond to the ones in (a). (c) Rank reordering file yielding the placement shown in (a).

placement based on the hardware resources available to the job at the time of job launching. Use of this placement style requires setting the MPICH_RANK_REORDER_OPTS environment variable which is explained in the mpi man page. The custom rank reordering option 3 allows the specification of a file MPICH_RANK_ORDER containing a permutation of $\{0, \ldots, N_P - 1\}$. The rows are comma-separated and the file is parsed from left to right and top to bottom. The MPI ranks of the application are placed in this order on the compute nodes. An example is shown in Figure 1 (c). In the simplified case of a two-dimensional nearest neighbor point-to-point communication pattern, a code can logically organize the ranks in a column-major order internally, as shown in Figure 1 (a), while ranks are physically placed by default in SMP-style, as shown in Figure 1 (b). In this scenario, a third of the off-node traffic, which is considerably more expensive than on-node communication, can be saved with a custom rank reordering such as in Figure 1 (a) using the rank reorder file shown in Figure 1 (c) which restored consistency between physical and logical rank placement. The MPICH_RANK_ORDER can be generated manually in case of small N_P or by means of the grid_order tool on Cray systems. This utility creates a rank order list for an MPI application that uses communication between nearest neighbors in a grid. The main command line arguments are the global grid dimensions (-g N1,N2...) and the dimensions of the cells into which the grid is subdivided (-c n1,n2...). In addition, one can specify if the grid follows a column-major (-C) of row-major (-R) ordering. This notation covers two and three-dimensional cases. It is preferable, but not required, that each cell dimension evenly divides the corresponding grid dimension, and that the product of the cell dimensions is equal to the number ranks that will be placed on a node. For the two-dimensional case described at the beginning of this section one has N1 $= P$, N2 $= Q$, n1 $= m_c$, and n2 $= n_c$ All rank numbers that fall into a given cell will be listed consecutively in the rank order that is produced and thus placed on the same

compute node. Note that the `MPICH_RANK_ORDER` only specifies which ranks are grouped together on a compute node but not on which particular node. The impact of specific node placement is considered to be negligible for the application in this work because the limiting factor is the injection bandwidth of a compute node. And especially on a dedicated Cray XC system, the inter node traffic is not shared. Setting the `MPICH_RANK_REORDER_DISPLAY` environment variable could be used to display which specific node each MPI rank resides in.

Given the vast variety of point-to-point communication patterns, the default SMP-style placement is usually not much worse than other schemes in terms of off-node communication for average workloads. However, in case of strong scaling experimental situations, opportunities to overlap regular point-to-point communications may not be available, which may engender performance penalties. Therefore, it becomes paramount to further reduce the network traffic with an appropriate custom rank reordering and load balancing for fast time to solution.

III. Polar Decomposition

A. Introduction

The polar decomposition (PD) for dense matrices is a major decomposition used in many applications including aerospace computations [3] and chemistry [4]. More recently, it has been used as a building block toward computing the singular value decomposition (SVD) and the symmetric eigenvalue decomposition (SEVD) [5]. The QDWH-PD algorithm is a backward stable iterative method, composed by successive calls to highly parallel compute-bound matrix operations (e.g., QR, Cholesky, matrix-matrix multiplication). More details can be found in [5], [6].

The polar decomposition of the matrix $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$ is written as $A = U_p H$. U_p is the polar factor, an orthogonal matrix obtained form the QDWH iteration procedure. $H = \sqrt{A^\top A}$ is a symmetric positive semidefinite matrix, from which a QDWH-based SEVD or SVD may be calculated recursively or directly, respectively.

B. The QDWH-based Polar Decomposition

The inverse-free QDWH-based iterative procedure computes the polar decomposition as follows [5], [7]:

$$\begin{aligned} X_0 &= A/\alpha, \\ \begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} &= \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \\ X_{k+1} &= \frac{b_k}{c_k} X_k + \frac{1}{\sqrt{c_k}} \left(a_k - \frac{b_k}{c_k} \right) Q_1 Q_2^\top, \ k \geq 0. \end{aligned} \tag{1}$$

When the matrix A is well-conditioned or its condition number gets improved after k iterations, it is possible to replace Eq. (1) with a Cholesky-based implementation as follows:

$$\begin{aligned} X_{k+1} &= \frac{b_k}{c_k} X_k + \left(a_k - \frac{b_k}{c_k} \right) (X_k W_k^{-1}) W_k^{-\top}, \\ W_k &= \text{chol}(Z_k), \ Z_k = I + c_k X_k^\top X_k. \end{aligned} \tag{2}$$

This further reduces the algorithmic complexity and, therefore, may speed up the overall computation [5].

While such a compute-intensive dense linear algebra algorithm is able to usually extract a good fraction of the system's theoretical peak performance, it may naturally lead to performance degradation in strong scaling experimental situations, when PEs run out of work. While this is expected for most DLA workloads, QDWH may be more sensitive to these challenging situations, due to its large number of successive DLA matrix operations. We propose to overcome these challenges using two approaches: a rank reordering technique combined with a grid topology strategy or an algorithmic paradigm shift based on the Zolotarev rational function. The former does not necessitate any changes on QDWH, while the latter revisits the QDWH core algorithm and introduces another level of parallelism at the expenses of extra floating-point operations (flops).

C. The ZOLO-PD Polar Decomposition

The main idea behind the ZOLO-PD algorithm is to generalize the rational approximant underlying the QDWH iterations, which results in the following iteration:

$$\begin{aligned} \begin{bmatrix} X \\ \sqrt{c_{2j-1}} I \end{bmatrix} &= \begin{bmatrix} Q_{j1} \\ Q_{j2} \end{bmatrix} R_j, \\ Z_{2r+1}(X; \ell) &= X + \Sigma_{j=1}^{s} \frac{a_j}{\sqrt{c_{2j-1}}} Q_{j1} Q_{j2}^*. \end{aligned} \tag{3}$$

Equation (3) exemplifies r embarrassingly parallel QR factorizations and matrix-matrix multiplications $Q_{j1} Q_{j2}^*$. Typically, for ill-conditioned matrices in double precision floating-point arithmetic, $s = 8$ and the number of iterations to converge is two. More analysis can be found in [6] and [8]. Similar to the QDWH algorithm, once X_k is well-conditioned, the QR-based iterations in Equation (3) can be replaced with Cholesky-based iterations with a lower arithmetic cost (see Equation (2)).

D. Algorithmic Complexity and Memory Footprint

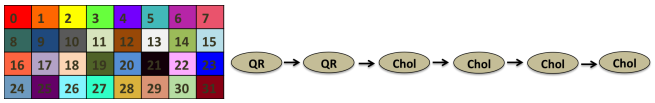
For ill-conditioned matrices with condition number $\kappa = 10^{12}$, QDWH performs two QR-based iterations followed by four Cholesky-based iterations. ZOLO-PD requires only two successive iterations but with many more operations per iteration, though embarrassingly parallel ones (up to $s = 8$ independent subproblems per iteration), as highlighted in [8]. In fact, the polar decomposition based on QDWH or ZOLO iterations reveals the existing trade-off between concurrency, algorithmic complexity and memory footprint.

The independent execution within ZOLO-PD iterations requires as many distinct data structures as the number of the parallel subproblems. Table I compares the resulting flop count and memory footprint of QDWH and ZOLO-PD for matrices with $\kappa = 10^{12}$. If the high concurrency of ZOLO-PD is not exploited, a single data structure can be reused for the independent subproblems as well as the subsequent iterations. However, the algorithmic complexity is expensive since the extra flops are computed sequentially. On the opposite, assuming there are enough PEs to solve each subproblem in parallel, the algorithmic complexity drops at the expense of an increase of the memory footprint to separately store each subproblem. ZOLO-PD performs then less than half of the QDWH operations.

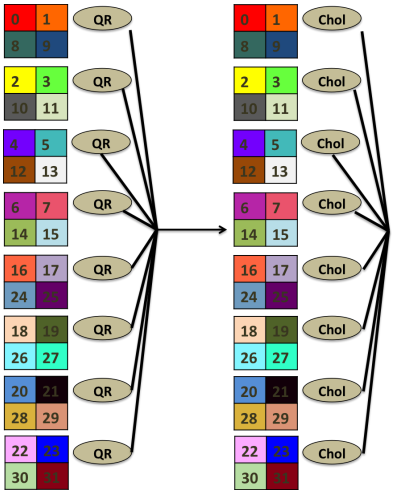
Table I: Algorithmic complexity and memory footprint for various PD algorithms with $\kappa_2(A) = 10^{12}$.

	QDWH	Successive ZOLO-PD	Independent ZOLO-PD
# QR-based iterations	2	8	1
# Cholesky-based iterations	4	8	1
Algorithmic complexity	$33n^3$	$100n^3$	$15\ n^3$
Memory footprint	$6n^2$	$6n^2$	$48n^2$

Figure 2 further distinguishes the three algorithmic variants for PD. Indeed, Figure 2a shows the PD iterations for QDWH which are done successively, and therefore, all processes work together in computing the corresponding QR or Cholesky-based iterations (up to six all-in-all). in ZOLO-PD, although the PD iterations are also performed successively, the overall number of processes is split in process subgroups to work concurrently within each iteration, as depicted in Figure 2b.



(a) Process grid for QDWH applied at each successive iteration: QR (2) and Cholesky (4).



(b) Independent process grids (with $s = 8$) for ZOLO applied within each successive iteration: QR (1) and Cholesky (1).

Figure 2: Process Grids for QDWH and ZOLO-PD.

IV. Simulation Results

The test bed for the simulations consists of a dedicated Cray XC system featuring dual Intel Broadwell processor compute nodes with 128GB DDR4 memory each and running with Moab/Torque+ALPS. The number of cores and base clock frequencies are not uniform across compute nodes. Only 32 cores per node were used with a frequency capped to 2.1GHz for the experiments. This amounts to 1075.2 GFlops/s theoretical peak performance per node in double precision floating-point arithmetic. The codes were built with the Intel Compiler 17.0.1.132 and the corresponding MKL library for the basic linear algebra computations. The enabled *Hugepages* feature was used for the factorizations. The matrix sizes considered for the polar decomposition with QDWH and ZOLO-PD range from 71680 to 122880 in steps of 10240 and are factorized on 200, 400, and 800 compute nodes using one MPI rank per core, where the MPI ranks are arranged by `ScaLAPACK` in a row-major order on a $P \times Q$ grid. The different rank reordering schemes for the simulations are summarized in Table II.

Table II: Different rank reorderings for the same simulation grid $P \times Q$ with corresponding label. Scheme 0 corresponds to the default SMP-style ordering.

Command	Label
`<no reordering>`	0
`grid_order -R -c 8,4 -g P,Q`	1
`grid_order -R -c 4,8 -g P,Q`	2
`grid_order -C -c 8,4 -g P,Q`	3
`grid_order -C -c 4,8 -g P,Q`	4

Both row-major and column-major global rank reorderings have been considered with two different on-node orderings. For a given amount of compute nodes, and therefore total number of MPI ranks, only a few grid topologies $P \times Q$ are possible with a compatible rank reordering. Table III summarizes the combinations used for the QDHW algorithm. Similarly, Table IV summarizes the combinations of rank reorderings and grid topologies $P \times Q$ used for the ZOLO-PD algorithm. The eight subproblems are solved on a sub-grid $p \times q$ which represents an additional degree of freedom compared to QDWH. Again, only a few compatible combinations of P, Q, p, and q exist for a given total amount of MPI ranks. The complete set of results for the QDWH algorithm, i.e. solver time as a function of matrix size N for 800, 400, and 200 compute nodes, and different combinations of reorderings schemes as well as

Table III: Combinations of rank reordering strategies and processor grid topologies $P \times Q$ on 800, 400, and 200 compute nodes for the QDWH algorithm with corresponding labels. The numbers in the reorder column correspond to the labels from Table II.

Nodes	Ranks	P	Q	$R = P/Q$	Reorder	Label
800	25600	160	160	1	0	v1_0
					1	v1_1
					2	v1_2
					3	v1_3
					4	v1_4
		128	200	0.64	0	v2_0
					1	v2_1
					2	v2_2
					3	v2_3
					4	v2_4
		100	256	0.39	0	v3_2
					2	v3_3
					4	v3_4
		50	512	0.098	0	v4_0
400	12800	100	128	0.78	0	v1_0
					2	v1_2
					4	v1_4
		80	160	0.5	0	v2_0
					1	v2_1
					2	v2_2
					3	v2_3
					4	v2_4
		64	200	0.3	0	v3_0
					1	v3_1
					2	v3_2
					3	v3_3
					4	v3_4
		32	400	0.08	0	v4_0
					1	v4_1
					2	v4_2
					3	v4_3
					4	v4_4
200	6400	80	80	1	0	v1_0
					1	v1_1
					2	v1_2
					3	v1_3
					4	v1_4
		64	100	0.64	0	v2_0
					1	v2_1
					3	v2_2
		32	200	0.16	0	v3_0
					1	v3_1
					2	v3_2
					3	v3_3
					4	v3_4

grid topologies is given in Figure 5. And the full set of results for the ZOLO-PD algorithm is shown in Figure 6. In particular, Figure 3 shows the QDWH and ZOLO-PD solver time for the largest matrix with $N = 122880$ as a function of the ratio $R = P/Q$, and $r = p/q$ for ZOLO-PD, for different reordering strategies. Sukkari et al. [9] observed an improvement of the total execution time for the QDWH algorithm when lowering the ratio R without reordering and this effect is also visible here over a wider range of R. Using a rank reordering which is consistent with the `ScaLAPACK` grid topology layout, i.e., row-major in this case, is more beneficial than column-major especially on a large amount of nodes, where network traffic starts to prevail over computation. Furthermore, the on-node reordering `-c 4,8` yields a considerable better performance than `-c 8,4` only for $R = 1$. When rank reordering is in use, the best performance is not achieved for large R but for an intermediate ratio. Figure 3 allows to identify the overall best combination of rank reordering and grid topology which never coincides with the default SMP-style ordering. In the case of the largest matrix on 800 nodes the combination $P = 128$, $Q = 200$, and reorder #1 yields an improvement of 54% over the SPM-style ordering on a square processor grid. This is also the best combination for the smallest matrix which achieves an improvement of 58% but is not shown here. Similarly, an optimal combination of reordering and topology can be identified for smaller node counts which is also not discussed here. Choosing row-major (1 and 2), which is consistent with the `ScaLAPACK` ordering, instead of column-major (3 and 4) rank reordering yields a smaller difference in performance for the square topology compared to the rectangular cases as shown in Figure 3. On the other hand, the ZOLO-PD algorithm does not manifest a comparable improvement when rank reordering or different grid topologies are used as shown in Figure 3. This is due to the overwhelming amount of extra flops required by ZOLO-PD, which permits to mitigate the communication overhead. However, strong scaling behavior is improved in both cases when using the best combination instead of the least performant SMP-style ordering as can be seen in Figure 4. The best solver times approach the ideal scaling curve of the least performant SMP-style ordering in the strong scaling limit.

V. Profiling Analysis

The impact of rank reordering and variation of grid topology on the QDWH polar decomposition of the largest matrix $N = 122880$ on 800 nodes is investigated by means of profiling analysis. In particular, the square ($R = 1$) and rectangular ($R = 0.64$) grid topology as well as the best performing combination of reordering and topology, i.e. reorder1 and $R = 0.64$. Lower node counts as well as the ZOLO-PD algorithm will not be considered in this section. Cray performance analysis tools are used to measure individual algorithm components and separate computation and communication time. Table V shows timings for the Cholesky and QR decomposition as well as the time spent for the estimation of condition number (timeLi) and the formation of the polar factor (timeFormH) for the three cases of interest relative to the total solver time and relative to the corresponding data for the square topology without rank reordering. Changing the grid topology from square to rectangular without additional reordering notably improves the QR and Cholesky decompositions while shifting the focus of the work towards QR. Using the most performant

Table IV: Combinations of rank reordering strategies and global processor grid topologies $P \times Q$ as well as the sub-grid layouts $p \times q$ for the eight subproblems on 800, 400, and 200 compute nodes for the ZOLO-PD algorithm with corresponding labels. The numbers in the reorder column correspond to the labels from Table II.

Nodes	Ranks	P	Q	p	q	$R = P/Q$	$r = p/q$	Reorder	Label
800	25600	80	320	40	80	0.25	0.5	1	v1_0
								1	v1_1
								2	v1_2
								3	v1_3
								4	v1_4
		80	320	20	160	0.25	0.125	0	v2_0
								2	v2_2
								4	v2_4
		160	160	40	80	1	0.5	0	v3_0
								1	v3_1
								2	v3_2
								3	v3_3
								4	v3_4
400	12800	80	160	40	40	0.5	1	0	v1_0
								1	v1_1
								2	v1_2
								3	v1_3
								4	v1_4
		64	200	32	50	0.32	0.64	0	v2_0
200	6400	40	160	20	40	0.25	0.5	0	v1_0
								2	v1_2
								4	v1_4
		40	160	10	80	0.25	0.125	0	v2_4
		80	80	20	40	1	0.5	0	v3_0
								2	v3_2
								4	v3_4

Table V: Time for individual components of the QDWH algorithm expressed as percentages of total solver time for the largest matrix $N = 122880$ on 800 nodes for two regular cases without rank reordering ($R \in \{1, 0.64\}$) and the optimal case with reorder1 and $R = 0.64$. The numbers in brackets in third and fourth columns are percentages of the total time of the regular case with $R = 1$.

	R=1	R=0.64	reorder1, R=0.64
Total	100	100 (75.44)	100 (47.96)
Cholesky	49.29	34.39 (25.94)	41.84 (20.06)
QR	40.53	52.92 (39.92)	46.81 (22.45)
timeLi	6.97	8.99 (6.79)	5.44 (2.61)
timeFormH	2.61	2.88 (2.18)	4.19 (2.01)

Table VI: Total MPI time and individual components expressed as percentages of total solver time for the largest matrix $N = 122880$ on 800 nodes for two regular cases without rank reordering ($R \in \{1, 0.64\}$) and the optimal case with reorder1 and $R = 0.64$. The numbers in brackets in third and fourth columns are percentages of the total time of the regular case with $R = 1$.

	R=1	R=0.64	reorder1, R=0.64
Total	100	100 (75.44)	100 (47.96)
Total MPI	84.07	76.56 (57.75)	60.26 (28.9)
Recv	53.33	44.08 (33.25)	25.89 (12.42)
Bcast	3.44	3.34 (2.52)	3.32 (1.59)
Bcast(sync)	3.81	4.95 (3.73)	5.98 (2.87)
Reduce	9.55	8.18 (6.17)	3.08 (1.48)
Reduce(sync)	10.51	12.05 (9.09)	16.86 (8.09)
Send	3.01	3.65 (2.75)	4.87 (2.34)

combination of rank reordering and grid topology further improves the individual decompositions and levels out the relative amount of work between the two. Table VI shows the total communication time and individual MPI components for the three cases of interest, again relative to the total solver time and relative to the corresponding data for the square topology without rank reordering. In all cases, a considerable amount of time is spent in communication with a dominating portion in `MPI_Recv` and collective synchronization in `MPI_Reduce`. Changing the grid topology from square to rectangular and then using the optimal combination of reordering and topology, successively reduces the relative amount of communication time while keeping the dominant portion of communication time in `MPI_Recv` and synchronization in `MPI_Reduce`. The gap between `MPI_Recv` and `MPI_Send` indicates that a large amount of time is spent in simply waiting in blocking receivers in point-to-point communication. Figure 7 shows the relevant part of the QDWH call-tree. The branches belonging to the QR and Cholesky factorizations are highlighted together with the associated `MPI_Recv` and `MPI_Reduce(sync)` leafs. The LU factorization `pdgetrf` is used for the estimation of the condition number and the top level `pdgemm` and `pdgeadd` for the formation of the polar factor. These branches also contain calls to `MPI_Recv` and `MPI_Reduce(sync)` but

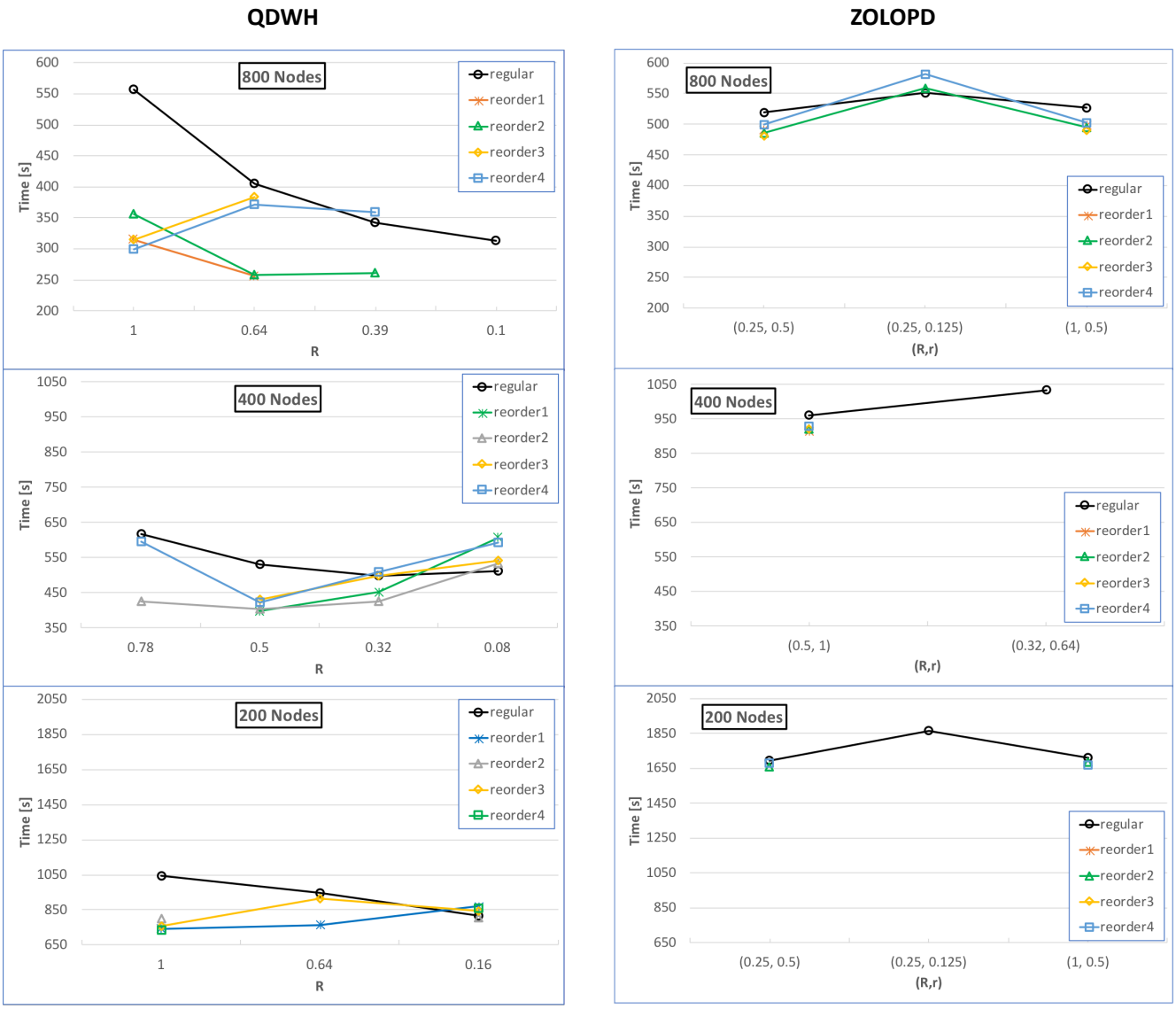


Figure 3: Left column shows the QDWH solver time as a function of the ratio $R = P/Q$ for the largest matrix size $N = 122880$ for different rank reordering strategies which are numbered according to Table II. Same for the ZOLO-PD algorithm in the right column with the addition of the ratio $r = p/q$ for the sub-problems as a parameter.

are less time consuming compared to QR and Cholesky in the main loop. As already mentioned in [9], the update of the trailing sub-matrix is the richest phase in terms of point-to-point communications for the Cholesky decomposition. Since more processors are involved in that phase when using a rectangular instead of a square topology, a higher degree of parallelism can be achieved. The resulting improvement in load balance helps improving the blocking time in point-to-point communication. This algorithmic improvement is solely attributed to the change in grid topology. However, modifying the topology also implies an improvement of the on-node communication as can be gathered from the traffic profiles. The rank reordering additionally reduces the off-node communication. Choosing row-major (1 and 2), which is consistent with the `ScaLAPACK` ordering, instead of column-major (3 and 4) rank reordering is expected to be more convenient because of row-wise and column-wise communication. And in the rectangular cases this is even more important. On the other hand, the ZOLO-PD algorithm benefits comparatively much less from reordering and different topologies. This confirms the early insights obtained from the performance results in Section IV. For the sizes considered herein, the workload per PE is significant and makes the application live in the compute-bound regime

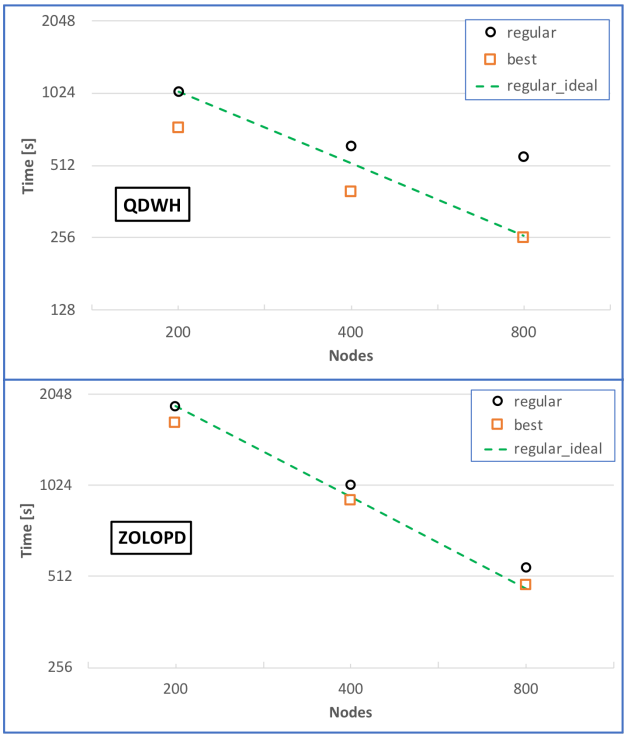


Figure 4: QDWH and ZOLO-PD solver time for the largest matrix $N = 122880$ as a function of the number of nodes. The best combination of rank reordering and grid topology is compared to the least performant (regular) SMP-style ordering. The dashed line shows the ideal strong scaling for the regular case.

throughout the execution. ZOLO-PD seems, therefore, to be more resilient to network traffic for the studied matrix sizes. Moreover, we have seen in [8] that ZOLO-PD may turn out to actually improve QDWH performance when running in strong scaling mode of operations for matrix sizes less than 70K. There have not been rank reorderings nor grid topology strategies employed in [8]. Applying both strategies for small matrix sizes should impact and further improve the performance of ZOLO-PD. The performance gap between ZOLO-PD and QDWH may then further reduce for the small matrix sizes, although we still anticipate ZOLO-PD to outperformance QDWH, since network congestion may ultimately impede QDWH performance more than ZOLO-PD.

VI. Integration in `cray-libsci`

The support for QDWH and its related KSVD singular value decomposition solver [9] was introduced in `cray-libsci/17.11.1` with updates to the API appearing in `cray-libsci/17.12.1`. Several environment variables appear in these releases to allow users to experiment with the convergence tolerance in the QDWH algorithm, to replace execution of `ScaLAPACK`'s `pdgesvd` with KSVD, and to use the ELPA eigenvalue solver library [10] as part of KSVD. For more information regarding these environment variables, the QDWH and KSVD API, and the most current features, one can refer to the `intro_qdwh` and `intro_ksvd` man pages in `cray-libsci`. The ZOLO-PD support is currently planned for future `cray-libsci` releases.

VII. Conclusion

We investigated the combined impact of grid topology and rank reordering on the polar decomposition of dense matrices using two advanced algorithms, i.e. QDWH and ZOLO-PD, building on top of `ScaLAPACK`. The focus is on medium to large matrix sizes, for which the strong scaling experiments may reach a limit, due to a possible dominant communication burden, despite dense linear algebra operations are naturally expected to be computation bound. An extensive number of simulations for several matrix sizes on different amount of nodes have been carried out on a dedicated Cray XC system. The QDWH algorithm profits considerably when choosing an appropriate combination of rank reordering and grid topology. An improvement of up to 54% in solver time on 800 nodes for the largest matrix could be observed. On the other hand, the ZOLO-PD algorithm benefits comparatively much less from reordering and different topologies, due to a higher computational workload per PE compared to QDWH. This makes ZOLO-PD less sensitive to network congestions. Though, strong scaling is still improved in both cases. A few interesting data points for the QDWH algorithm have been further analyzed by means of Cray performance tools. The QDWH algorithm is, as expected, clearly communication bound and most of the time is spent in the main loop computing QR or Cholesky decompositions. The dominant part of the communication time is spent in `MPI_Recv` which is successively reduced by using a rectangular topology and even further with the most performant combination of reordering and topology. The results achieved in this work are not necessarily directly transferable to other dense linear algebra algorithms based on `ScaLAPACK` but the present analysis should induce users to experiment with the rank reordering feature of `cray-mpich` and the grid topology strategies. This may render considerable performance improvement with a relatively low effort from the end-users' perspective.

Acknowledgment

The authors would like to thank Mohammed Hadi, Nick Hill and Luiz DeRose from the Cray LibSci developer team for their effort in testing and integrating QDWH software.

References

[1] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK users' guide*. Siam, 1979.

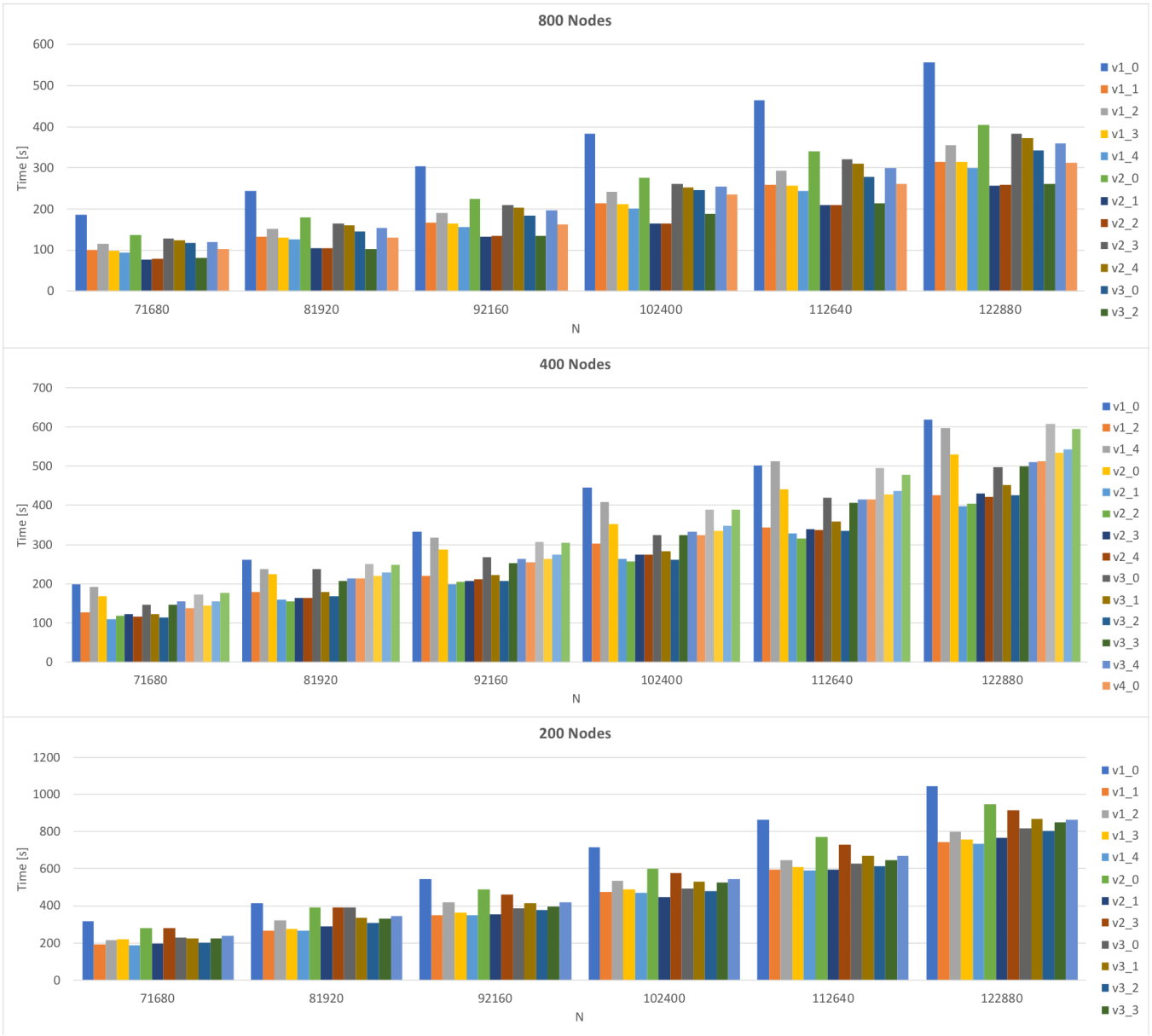


Figure 5: QDWH solver time as a function of matrix size N for 800, 400, and 200 compute nodes for different combinations of rank reorderings and grid topologies. The labels in the legend are described in Table III.

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

[3] I. Bar-Itzhack, "Iterative optimal orthogonalization of the strapdown matrix," *Aerospace and Electronic Systems, IEEE Trans. on*, vol. AES-11, no. 1, pp. 30–37, Jan 1975.

[4] J. A. Goldstein and M. Levy, "Linear algebra and quantum chemistry," *Am. Math. Monthly*, vol. 98, no. 10, pp. 710–718, Oct. 1991. [Online]. Available: http://dx.doi.org/10.2307/2324422

[5] Y. Nakatsukasa and N. J. Higham, "Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD," *SIAM Journal on Scientific Computing*, vol. 35, no. 3, pp. A1325–A1349, 2013.

[6] Y. Nakatsukasa and R. W. Freund, "Computing Fundamental Matrix Decompositions Accurately via the Matrix Sign Function in Two Iterations: The Power of Zolotarev's Functions," *SIAM Review*, vol. 58, no. 3, pp. 461–493, 2016. [Online]. Available: http://dx.doi.org/10.1137/140990334

[7] Y. Nakatsukasa, Z. Bai, and F. Gygi, "Optimizing Halley's Iteration for Computing the Matrix Polar Decomposition,"

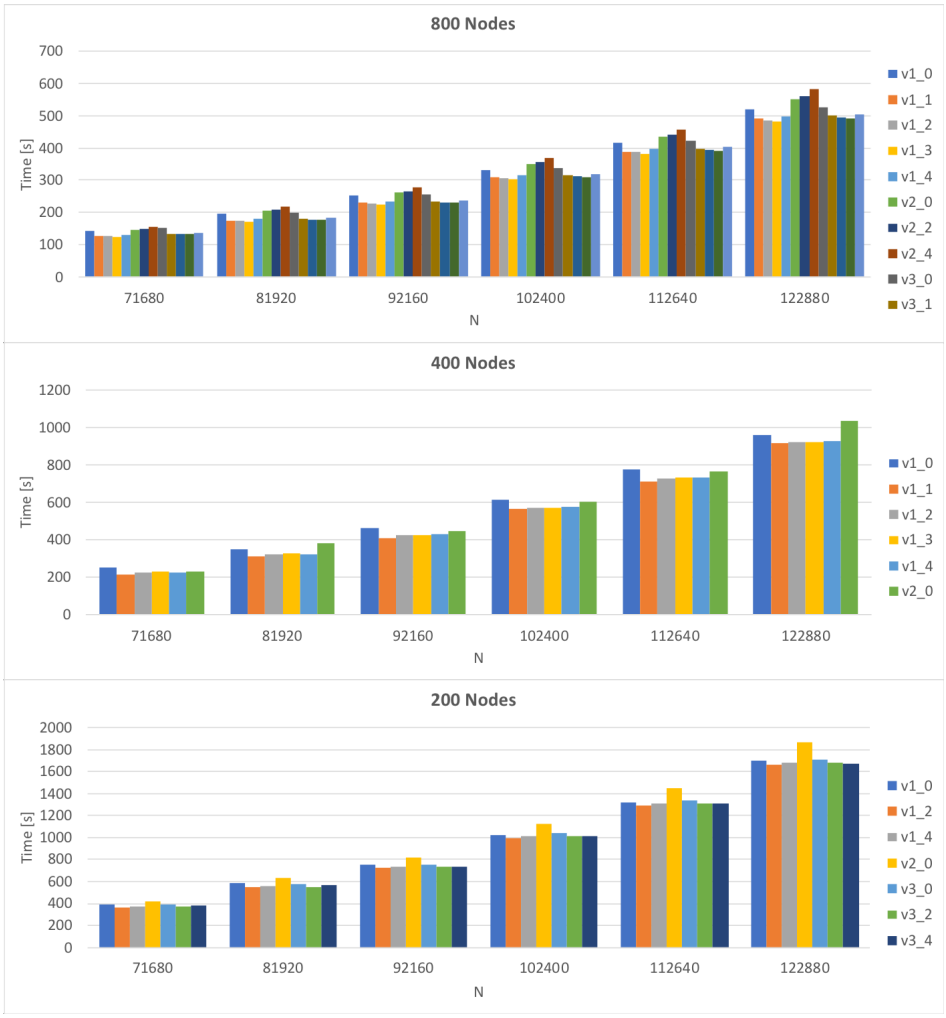


Figure 6: ZOLO-PD solver time as a function of matrix size N for 800, 400, and 200 compute nodes for different combinations of rank reorderings and grid topologies. The labels in the legend are described in Table IV.

SIAM Journal on Matrix Analysis and Applications, vol. 31, no. 5, pp. 2700–2720, 2010.

[8] H. Ltaief, D. Sukkari, A. Esposito, Y. Nakatsukasa, and D. Keyes, "Massively Parallel Polar Decomposition on Distributed-Memory Systems," *Submitted to ACM Transactions on Parallel Computing, available at http://hdl.handle.net/10754/626359*, 2018.

[9] D. Sukkari, H. Ltaief, A. Esposito, and D. Keyes, "A QDWH-Based SVD Software Framework on Distributed-Memory Manycore Systems," *Submitted to ACM Transactions on Mathematical Software, KAUST Technical Report Available at http://hdl.handle.net/10754/626212*, 2017.

[10] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H. Bungartz, and H. Lederer, "The ELPA Library: Scalable Parallel Eigenvalue Solutions for Electronic Structure Theory and Computational Science," *J Phys Condens Matter*, vol. 26, no. 21, 2014. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/24786764

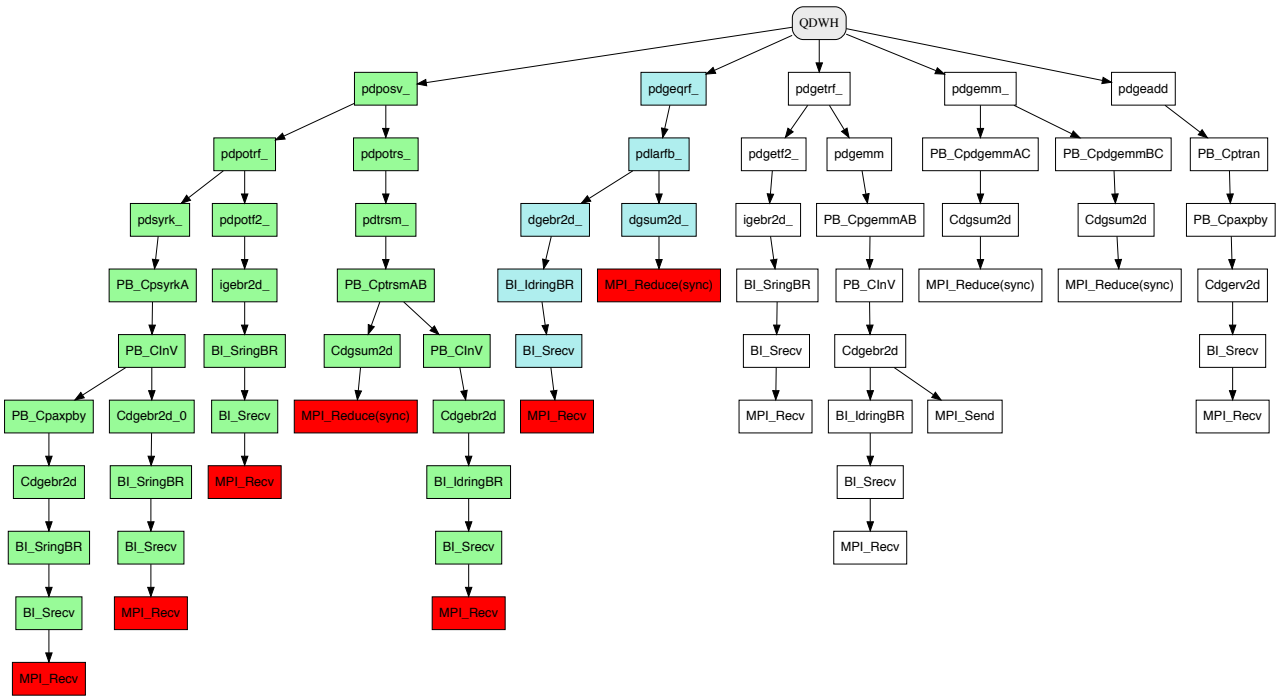


Figure 7: Call-tree of the QDWH algorithm highlighting the Cholesky (green) and QR (blue) component. The associated `MPI_Recv` and `MPI_Reduce(sync)` are marked in red. The other branches are the LU factorization (`pdgetrf`) used for the estimation of the condition number and `pdgemm` and `pdgeadd` for the formation of the polar factor.