A High Performance SVD Solver on Manycore Systems

Dalal Sukkari[1], Hatem Ltaief[1], Aniello Esposito[2], and David Keyes[1]

[1]KAUST Extreme Computing Research Center
[2]Cray EMEA Research Lab

Cray User Group Meeting
Redmond, WA USA

May 7-11, 2017

Outline

Outline

Main Component: Polar Decomposition

- The polar decomposition:

$$A \;=\; U_p H \;,\; A \in \mathbb{R}^{m\times n}(m \geq n)$$

  where, U_p is an orthogonal matrix and $H = \sqrt{A^\top A}$ is a symmetric positive semidefinite matrix
- Directly, using the singular value decomposition (SVD):

$$A \;=\; U\Sigma V^\top \;=\; (UV^\top)(V\Sigma V^\top) \;=\; U_p H$$

- Iteratively, using the inverse free QR dynamically-weighted Hally iteration (QDWH)

Application to EIG/SVD

- The polar decomposition is a critical numerical algorithm for various applications, including aerospace computations, chemistry, factor analysis
- The polar decomposition can be used as pre-processing step toward calculating the SVD
  $A = U_p H = U_p(V\Sigma V^\top) = (U_p V)\Sigma V^\top = U\Sigma V^\top$
- The polar decomposition can be used as pre-processing step toward calculating the EVD $A = V\Lambda V^\top$, $V = [V_1 V_2]$
  $A = U_p H$, then

$$
\begin{aligned}
U_p + I &= \begin{bmatrix} V_1 & V_2 \end{bmatrix} \begin{bmatrix} I_k & 0 \\ 0 & -I_{n-k} \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^* + I \\
&= \begin{bmatrix} V_1 & V_2 \end{bmatrix} \begin{bmatrix} I_k & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^* \\
&= 2V_1V_1^*
\end{aligned}
$$

Outline

Polar Decomposition (QDWH)

$$A = U_p H$$

where, $U_p U_p^\top = I_n$, H is symmetric positive semidefinite

- Backward stable algorithm for computing the polar decomposition
- Based on conventional computational kernels, i.e., Cholesky/QR factorizations (≤ 6 iterations for double precision) and GEMM
- The total flop count for QDWH depends on the condition number of the matrix κ:

κ	1	10^{16}
flops	$(10 + \frac{2}{3})n^3$	$43n^3$

Polar Decomposition QDWH (cont'd)

The QDWH iteration is:

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

When, X_k becomes well-conditioned, it is possible to replace Equation 1 with a Cholesky-based implementation as follows:

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

Polar Decomposition QDWH (cont'd)

$$\begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \; X_{k+1} = \frac{b_k}{c_k} X_k + \frac{1}{\sqrt{c_k}} (a_k - \frac{b_k}{c_k}) Q_1 Q_2^\top \tag{3}$$

where,

$$a_k = h(l_k), \; b_k = (a_k - 1)^2/4, \; c_k = a_k + b_k - 1$$

$$\sigma(X_k) = l_k = \frac{l_{k-1}(a_{k-1} + b_{k-1} l_{k-1}^2)}{1 + c_{k-1} l_{k-1}^2}, \; k = 1, 2, \ldots$$

$$h(l) = \sqrt{1+d} + \frac{1}{2}\sqrt{8 - 4d + \frac{8(2 - l^2)}{l^2\sqrt{1+d}}}, \; d = \sqrt[3]{\frac{4(1 - l^2)}{l^4}}$$

The convergence of the iterate X_{k+1} to the polar factor is measured by the closeness of its singular values $\sigma_i(X_{k+1})$ to 1. Hence, the number of QDWH iterations for convergence is the first k such that $|1 - l_k| < 10^{-16}$. Ex: $l_0 = 10^{-16}$

#it	$a_{\#it}$	$b_{\#it}$	$c_{\#it}$	$\sigma(X_{\#it})$
1	1.2×10^{11}	3.4×10^{21}	3.4×10^{21}	1.16×10^{-5}
2	4.5×10^{3}	5.9×10^{6}	5.8×10^{6}	0.057
3	17.09	64.7	80.8	0.78
4	3.4	1.4	3.8	0.89
5	3.0003	1.003	3.007	0.99
6	3	1	3	1.0

Outline

ScaLAPACK 101

We implement QDWH using the state-of-the-art vendor-optimized ScaLAPACK from the Cray Scientific library (libSCI):

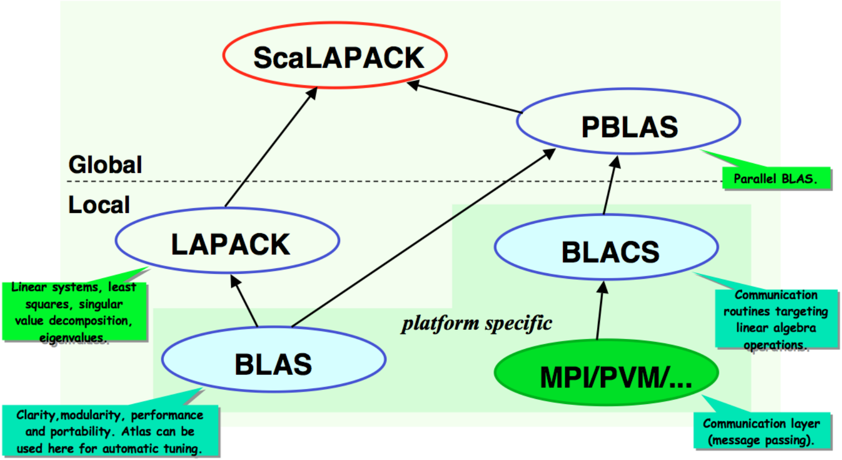
- Relies on block algorithms, similar to LAPACK
- Employs the bulk synchronous programming model
- Uses 2D block cyclic data distribution to map the matrix data to the distributed-memory
- Relies on the Message Passing Interface (MPI) to exchange data within the grid of MPI processes

ScaLAPACK 101

ScaLAPACK Block Algorithm:

- Panel-Update Sequence
- Transformations are blocked/accumulated within the Panel (Level 2 PBLAS)
- Transformations applied at once on the trailing submatrix (Level 3 PBLAS)
- Parallelism hidden inside the PBLAS

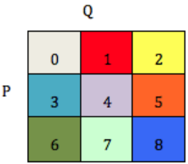
ScaLAPACK 101



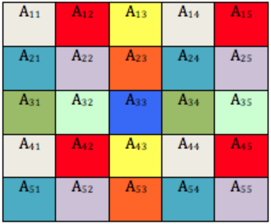
Source: *Short Course on the DOE ACTS Collection - SIAM CSE05 Conference Orlando, FL - February 11, 2005*

ScaLAPACK 101

2D Block-Cyclic Data Distribution:



Processes Grid



Logical View (Matrix)

Local View (CPUs)

Environment Settings

Software:

- Intel Compiler Suites v$15.0.1.133$
- Cray LibSci/13.2.0 ScaLAPACK library
- Square MPI process grid P=Q
- Block size nb $= 64$
- Ill and Well conditioned matrices generated using ScaLAPACK routine PDLATMS

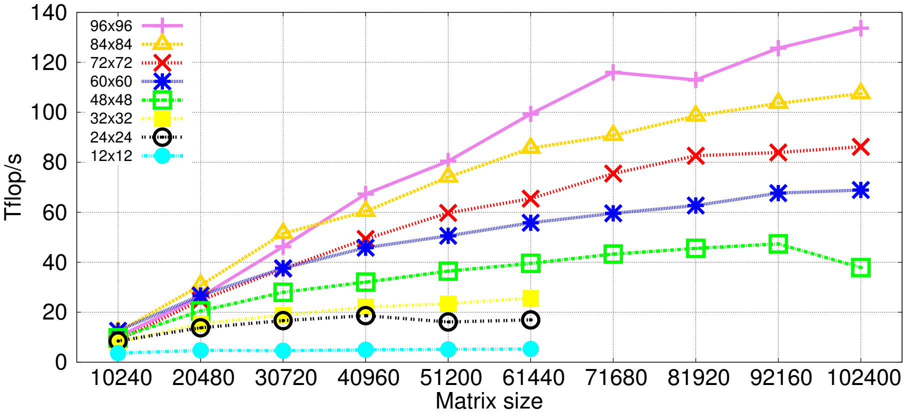
Hardware: The Cray XC40 system codenamed *Piz Dora* installed at the Swiss National Supercomputing Centre (CSCS). *Piz Dora* has 1256 compute nodes, each node has:

- Two-sockets Intel Xeon E5-2690 v3 (Haswell)
- 12 cores each running at 2.60GHz
- 64GB of DDR3 main memory
- The theoretical peak performance of the system is 1.254 Petaflops

Outline

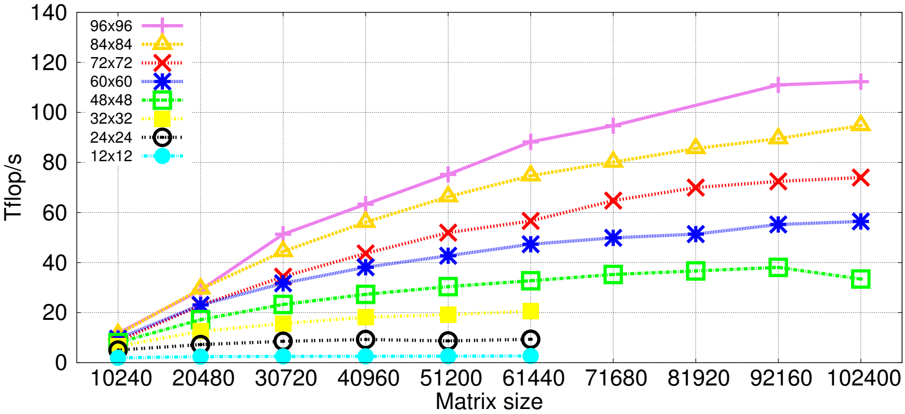
QDWH performance results in Tflop/s

Well-Conditioned Matrix



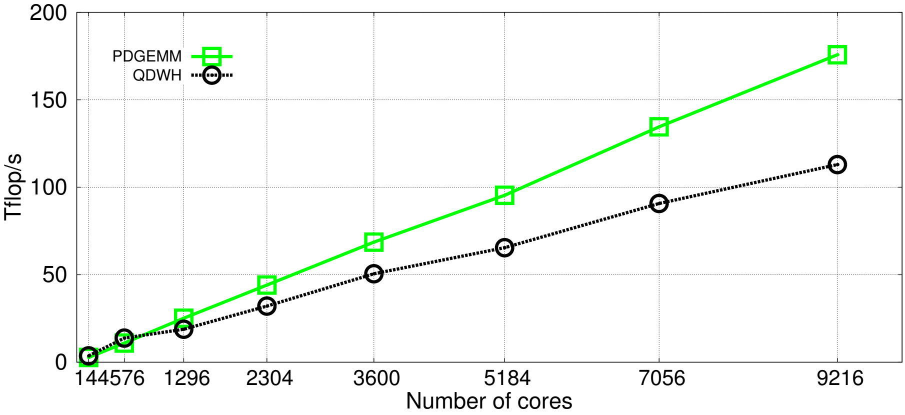
QDWH performance results in Tflop/s

Ill-Conditioned Matrix



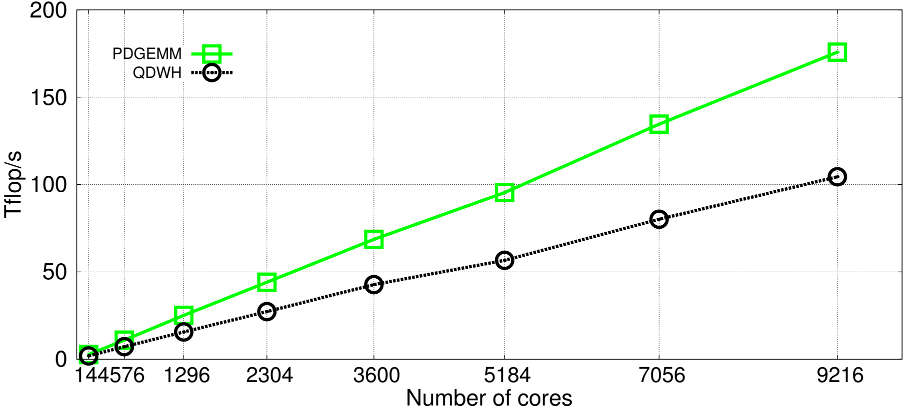
QDWH scalability assessment

Well-Conditioned Matrix



QDWH scalability assessment

Ill-Conditioned Matrix



Outline

QDWH-based SVD

Algorithm 1 QDWH-SVD [1]

Compute the polar decomposition $A = U_p H$ via QDWH
Compute the eigenvalue decomposition $H = V\Sigma V^\top$
Compute the right singular vectors $U = U_p V$
Hence the SVD $A = U\Sigma V^\top$

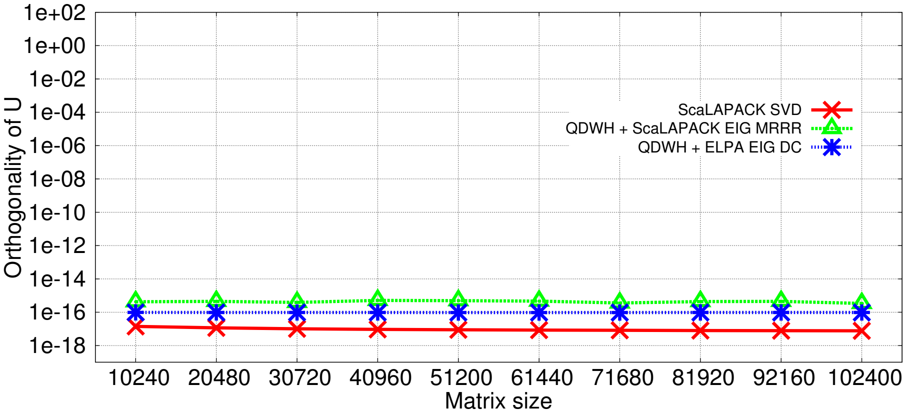
We used different eigensolvers to build QDWH-SVD:

- PDSYEVR ScaLAPACK eigensolver based on a 1-stage tridiagonal reduction and the MRRR eigensolver
- ELPA-EIG which combines a two-stage reduction with a divide-and-conquer eigensolver

[1] Yuji Nakatsukasa, Nicholas J. Higham: Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD. SIAM J. Scientific Computing 35(3) (2013)

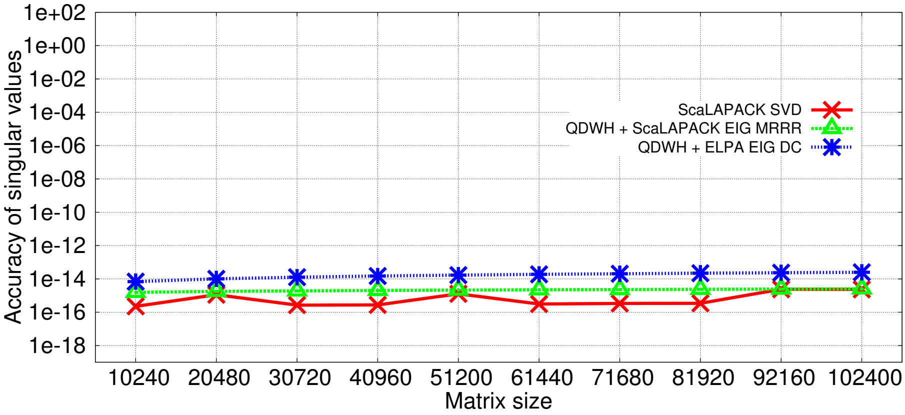
Orthogonality of U (similar to V)

Well-Conditioned Matrix



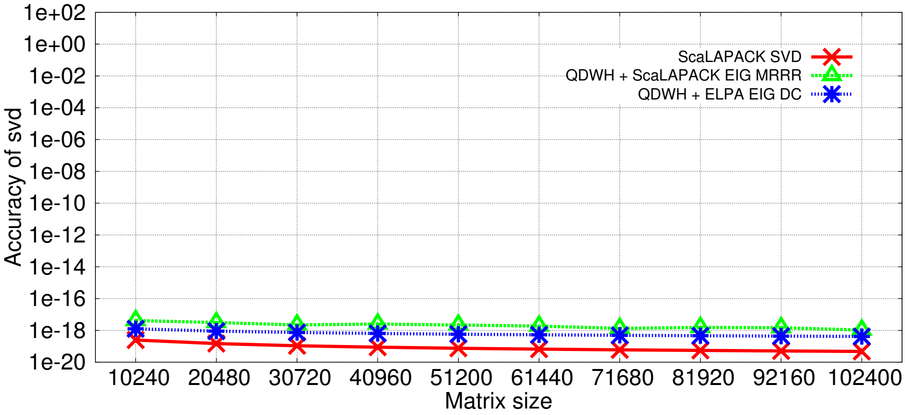
Accuracy of Singular Values

Well-Conditioned Matrix



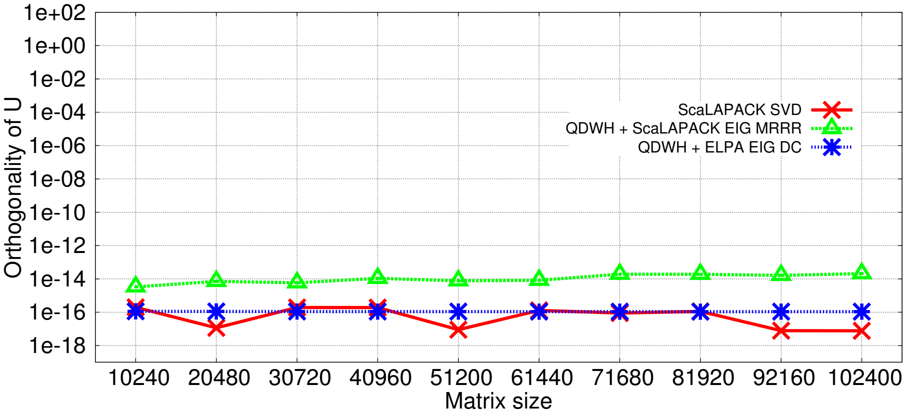
Accuracy of SVD

Well-Conditioned Matrix



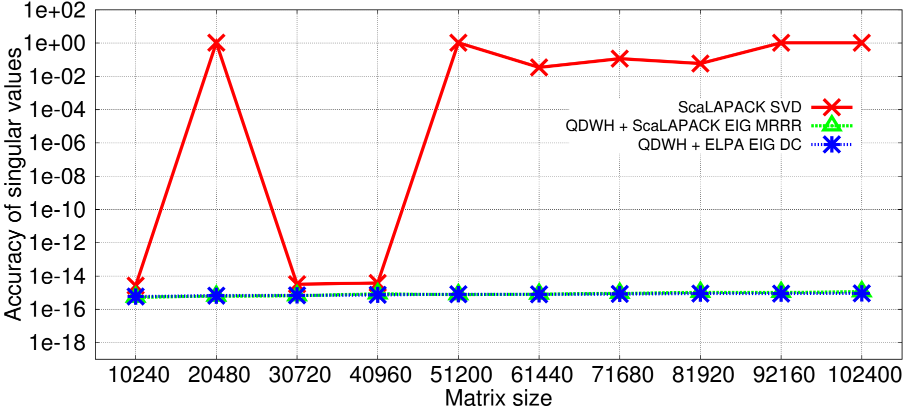
Orthogonality of U (similar to V)

Ill-Conditioned Matrix



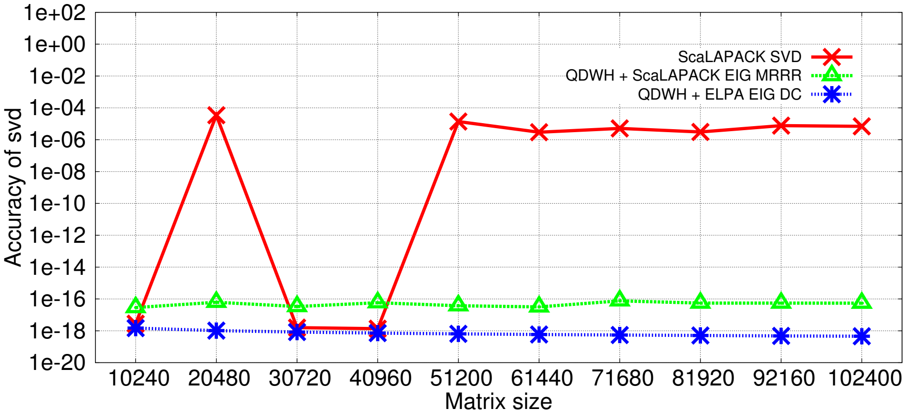
Accuracy of Singular Values

Ill-Conditioned Matrix

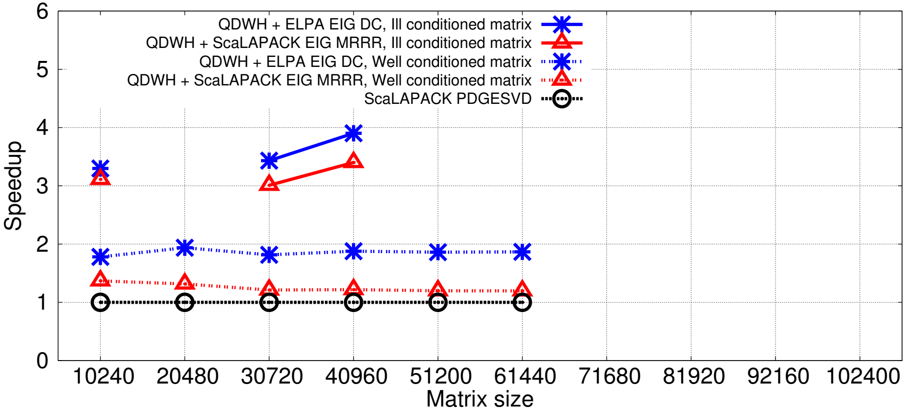


Accuracy of SVD

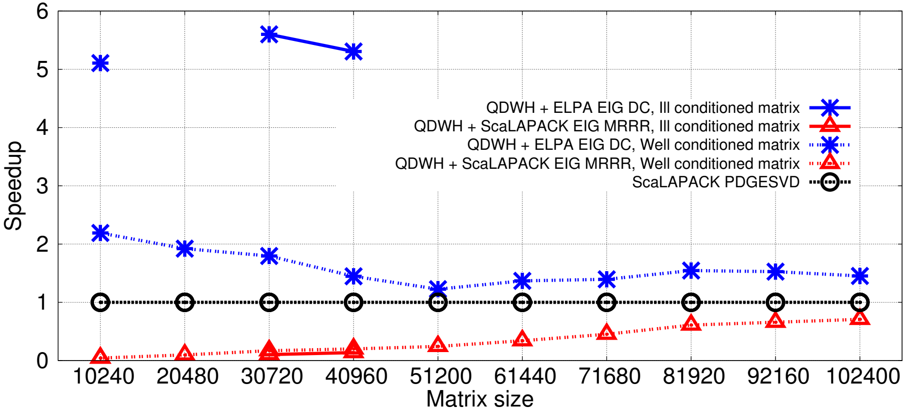
Ill-Conditioned Matrix



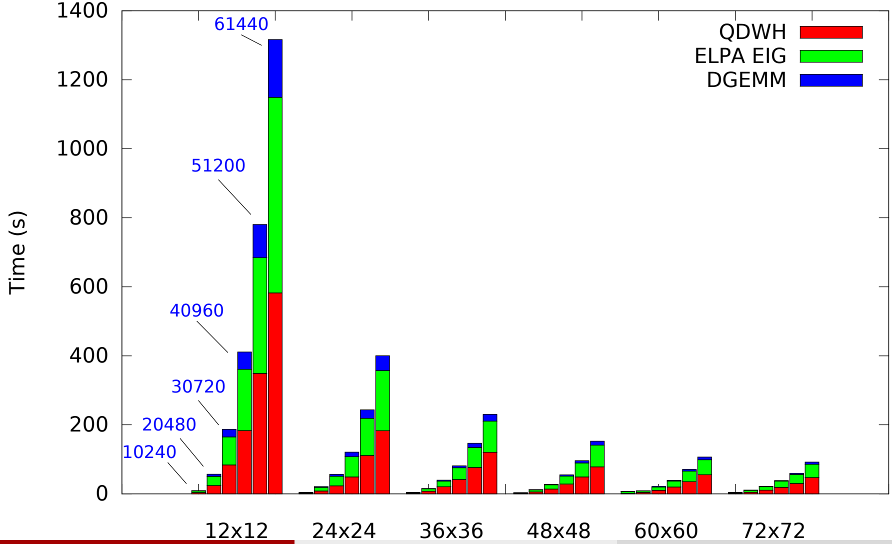
Performance comparison of SVD solvers in Time: 12x12



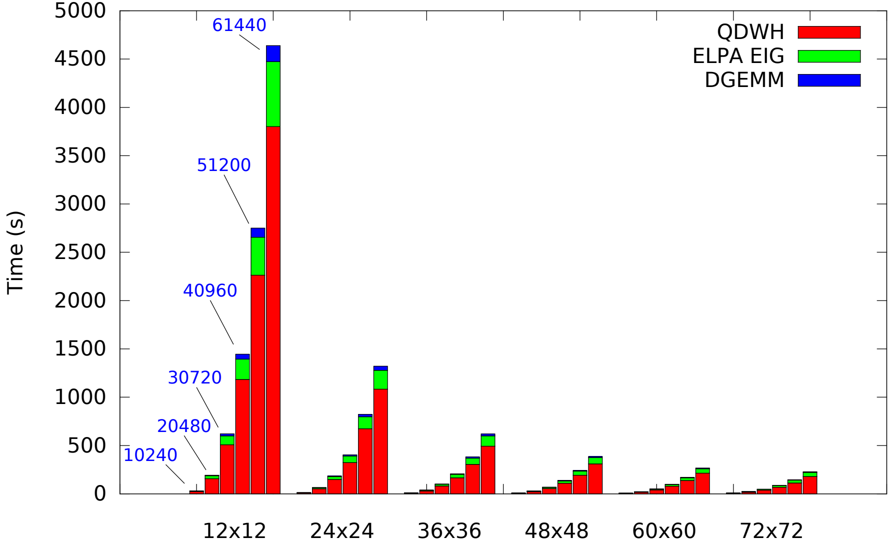
Performance comparison of SVD solvers in Time: 96x96



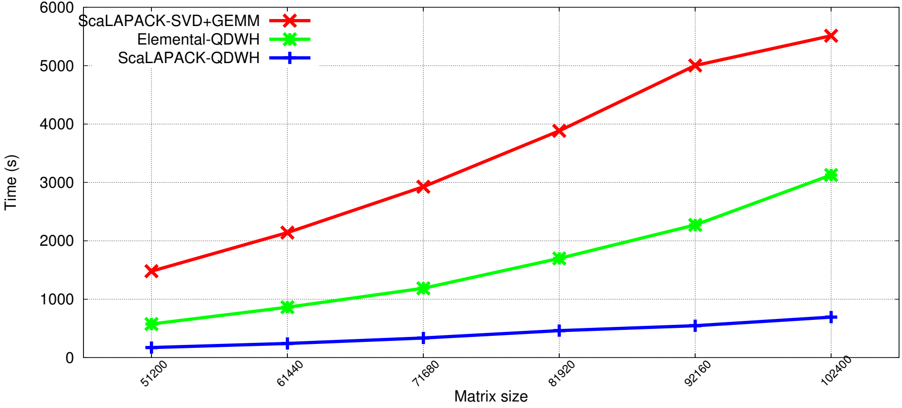
Profiling the Computational Stages of the ELPA-based QDWH-SVD: Well-Conditioned Matrix



Profiling the Computational Stages of the ELPA-based QDWH-SVD: Ill-Conditioned Matrix

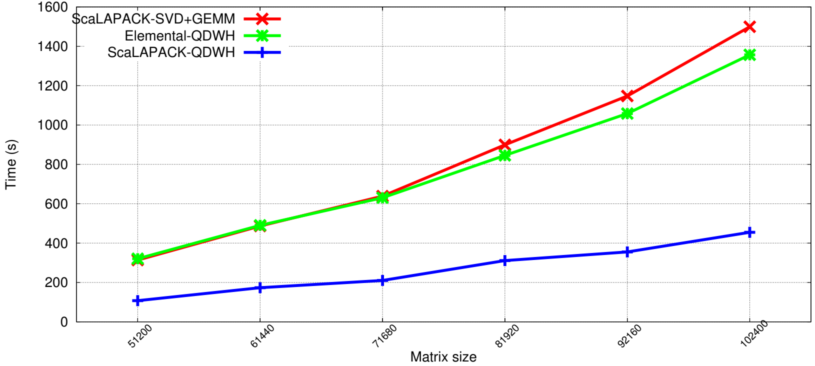


QDWH Performance on Cray KNL-based Distributed-Memory System (quadrant/cache)



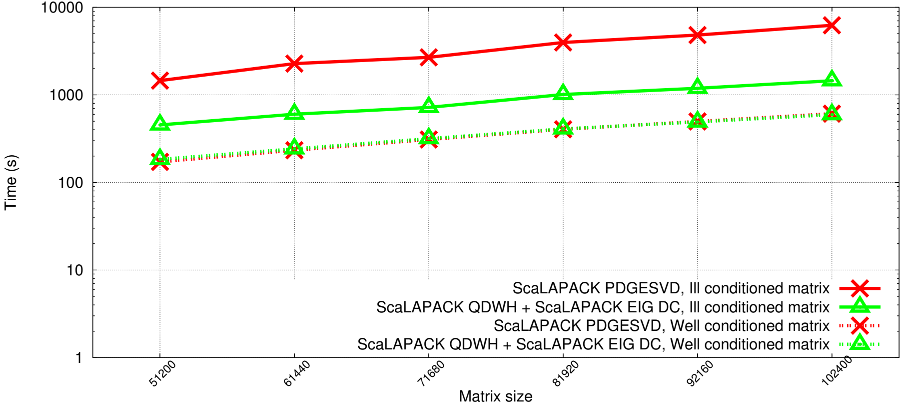
144 nodes, [P=64; Q=144], up to 8X speedup

QDWH Performance on Shaheen-2 Cray XC 40



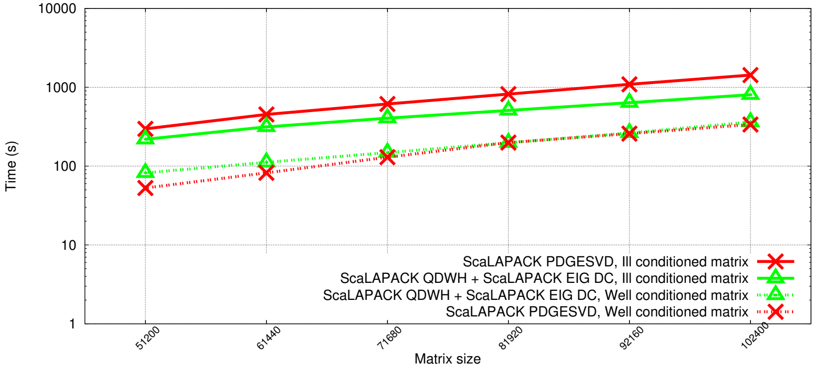
288 nodes, [P=64; Q=144], up to 3.5X speedup

QDWH-SVD Performance on Cray KNL-based Distributed-Memory System (quadrant/cache)



144 nodes, [P=64; Q=144], up to 4.3X speedup

QDWH-SVD Performance on Shaheen-2 Cray XC 40



288 nodes, [P=64; Q=144], up to 1.8X speedup

Outline

Related Work

- Higham and Papadimitriou (1994), matrix inversion QDWH, shared-memory systems
- Nakatsukasa et. al (2010), inverse-free QDWH, theoretical accuracy study
- Poulson et. al (2012), Elemental, distributed-memory system
- Nakatsukasa and Higham (2013), QDWH-EIG, QDWH-SVD, theoretical accuracy study
- Sukkari, Ltaief and Keyes (2016), QDWH-SVD, shared-memory system equipped with multiple GPUs

Outline

Conclusion and Future Work

- The performance analysis shows decent scalability running with around 9200 MPI processes on well and ill- conditioned matrices of 100K x 100K problem size
- The performance impact of using QDWH as a pre-processing step toward calculating the SVD achieves up to 4.3X speedup speedup against ScaLAPACK PDGESVD
- The numerical accuracy study highlights the robustness of QDWH-SVD over ScaLAPACK PDGESVD
- Bulk synchronization model
- Fork-join model
- Task-based asynchronous QDWH
- Implementation on shared and distributed memory
- Application to EVD and SVD

Thank you ☺