Designing Fast Symmetric Eigenvalue Solvers on Manycore Systems

Hatem Ltaief[1] Piotr Łuszczek[2] Dalal Sukkari[3]

Supercomputing Laboratory [1]
KAUST, Saudi Arabia

Innovative Computing Laboratory [2]
University of Tennessee Knoxville

Computer, Electrical, and Mathematical Sciences and Engineering Division [3]
KAUST, Saudi Arabia

SIAM Conference on Computational Science and Engineering
Feb 25-March 1, 2013

Outline

1. Stating the Problem
2. Standard Approach
3. PLASMA - Two-stage Approach
4. Other Approach: QDWH

Problem Definition

- Generalized eigenvalue problem:

$$Ax = \lambda Bx,$$

$$A, B \in \mathbb{R}^{n \times n},\ x \in \mathbb{R}^n,\ \lambda \in \mathbb{R}$$

with A being a symmetric or Hermitian matrix ($A = A^T$ or $A = A^H$),
B being a symmetric or Hermitian positive definite matrix,
λ – an eigenvalue,
and x the corresponding eigenvector.
Solved by first transforming the problem to a standard eigenvalue problem (Chol(B) + applying inverted factors).

- Standard eigenvalue problem:

$$B = I_n$$

Problem Definition

- Generalized eigenvalue problem:

$$Ax = \lambda Bx,$$

$$A, B \in \mathbb{R}^{n \times n},\ x \in \mathbb{R}^n,\ \lambda \in \mathbb{R}$$

with A being a symmetric or Hermitian matrix ($A = A^T$ or $A = A^H$),
B being a symmetric or Hermitian positive definite matrix,
λ – an eigenvalue,
and x the corresponding eigenvector.
Solved by first transforming the problem to a standard eigenvalue problem (Chol(B) + applying inverted factors).

- Standard eigenvalue problem:

$$B = \mathrm{I}_n$$

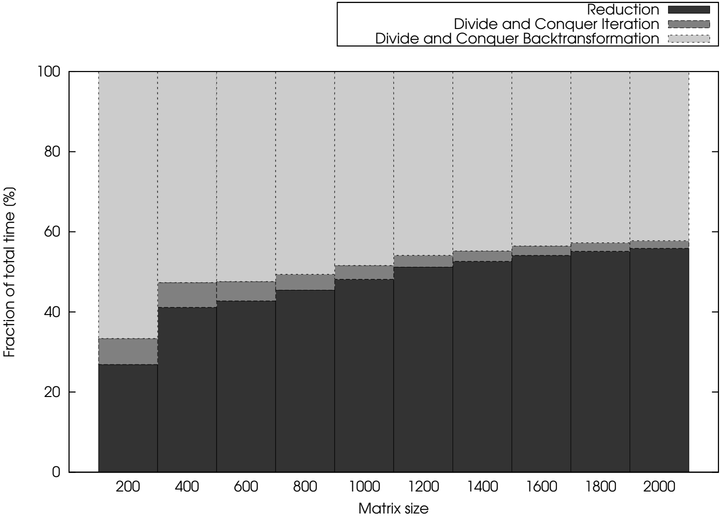
LAPACK - Standard Approach

The goal is to transform the matrix A into a symmetric (or Hermitian) tridiagonal matrix T:

$$T = Q\,A\,Q^T,$$

$$A, Q, T \in \mathbb{R}^{n \times n}.$$

Time Breakdown: TRD, Eig-Values, and Eig-Vectors

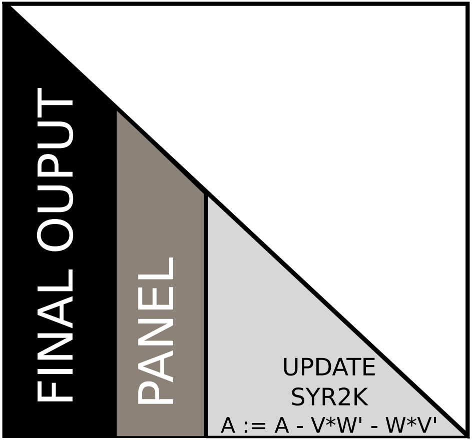


LAPACK - Block Algorithms

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

LAPACK - Block Algorithms

- Panel computation involves the entire trailing submatrix
- Performance are impeded by memory-bound nature of the panel
- Reductions achieved through one-stage approach



LAPACK - Block Algorithms

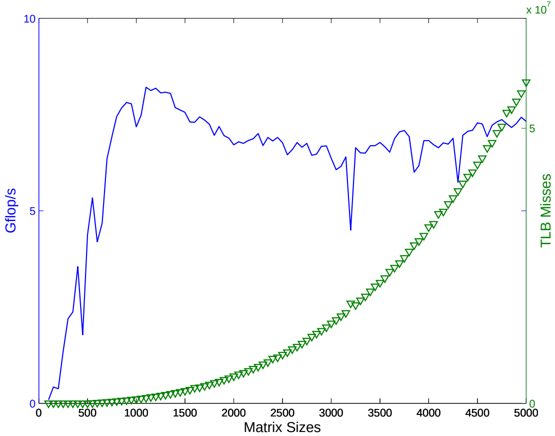


Figure : Performance evaluation and TLB miss analysis of the one-stage LAPACK TRD algorithm with optimized Intel MKL BLAS, on a dual-socket quad-core Intel Xeon (8 cores total).

PLASMA - Tile Algorithms

- Parallelism is brought to the fore
- Tile data layout translation
- May require the redesign of linear algebra algorithms
- Remove unnecessary synchronization points between Panel-Update sequences
- DAG execution where nodes represent tasks and edges define dependencies between them
- Dynamic runtime system environment

PLASMA - Data Layout Translation

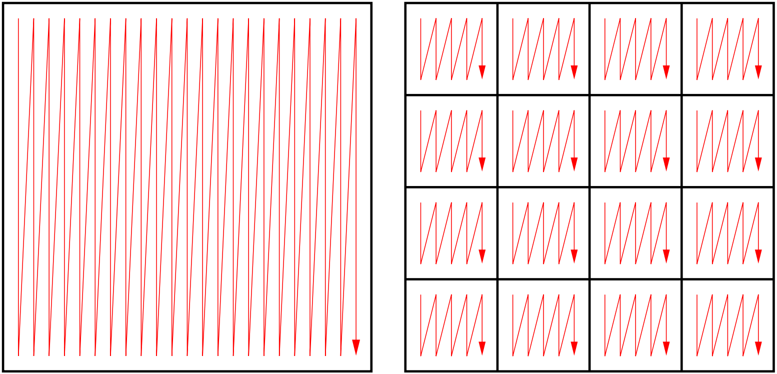


Figure : Translation from LAPACK Layout (column-major) to Tile Data Layout

PLASMA - Two-stage

C. Bischof, B. Lang and X. Sun, **Successive Band Reductions**, ACM TOMS, 2000.
The accompanying software is called `SBR Toolkit`.

Stage I: Band Reduction

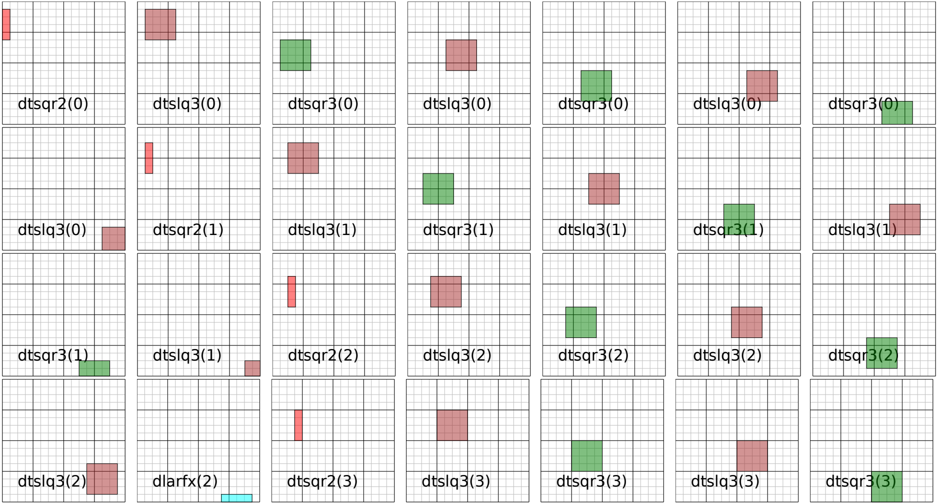
- Tile algorithm running on top of tile data layout
- Rely on high performant compute-intensive kernels
- Composed by successive calls to Level 3 BLAS operations
- Derived from QR factorization kernels
- Handle cautiously the symmetric structure of the matrix

PLASMA - Two-stage

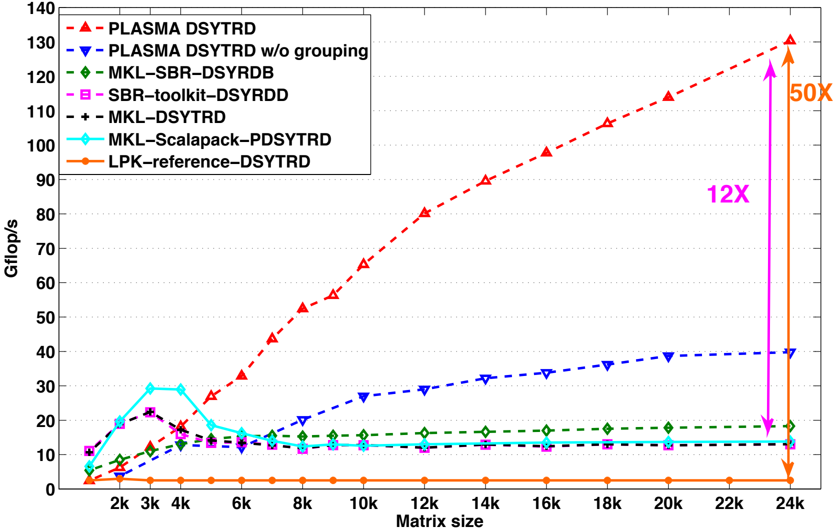
Stage II: Bulge Chasing

- Further reduce the band tridiagonal matrix to the final tridiagonal form
- Algorithm proceeds by column-wise annihilation
- Each column annihilation (or sweep) creates a bulge, which needs to be chased down to the bottom right corner of the matrix
- If N is the matrix size, $(N - 2)$ sweeps are required to achieve the tridiagonal structure.
- Rely on Level 2 BLAS kernels
- Highly memory-bound operations: the whole matrix needs to be traversed to annihilate a single column.

PLASMA - Two-stage: Stage II Bulge Chasing

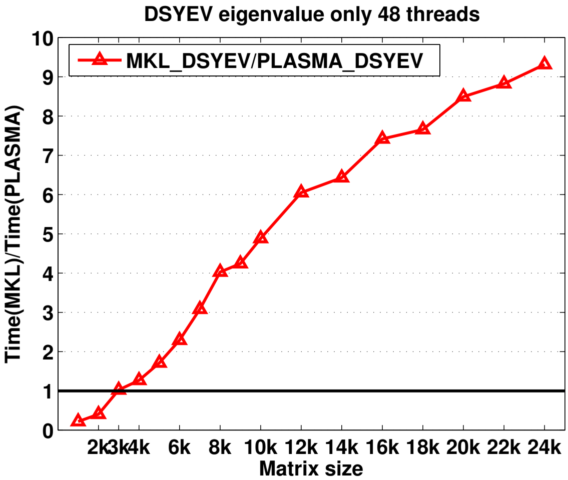


PLASMA - Two-stage: TRD



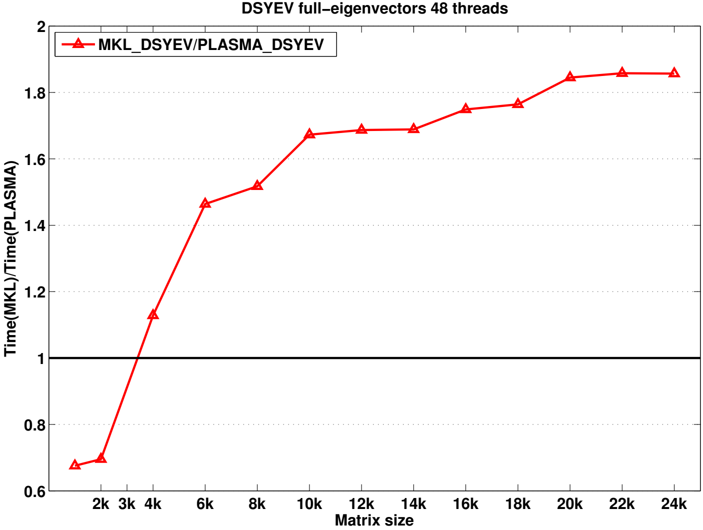
Azzam Haidar, Hatem Ltaief and Jack Dongarra, SC'11

PLASMA - Two-stage: TRD + Eigenvalue

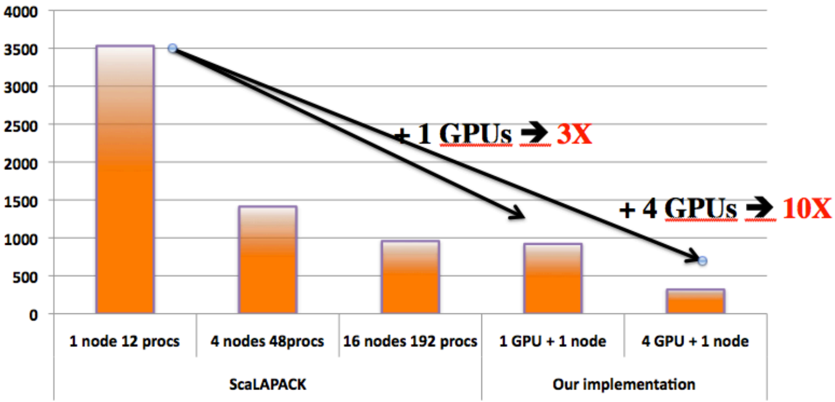


Azzam Haidar, Hatem Ltaief and Jack Dongarra, SC'11

PLASMA - Two-stage: TRD + Eigenvalue + Eigenvector

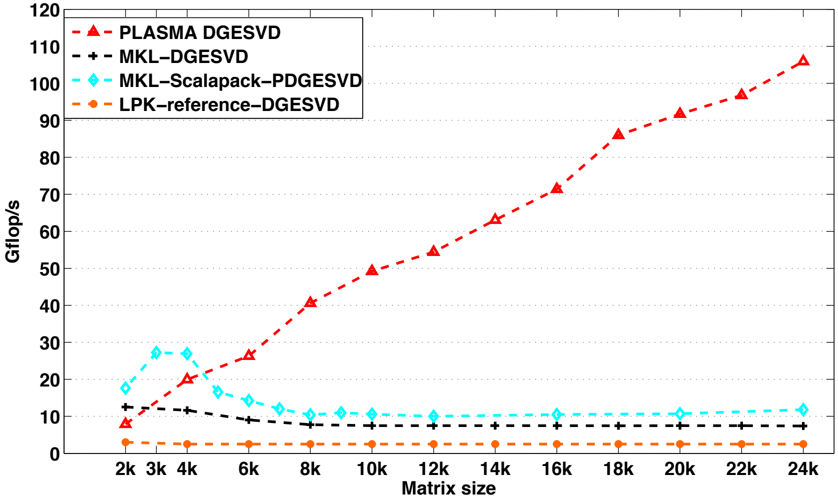


PLASMA - Two-stage: TRD + Eigenvalue + Eigenvector + GPU



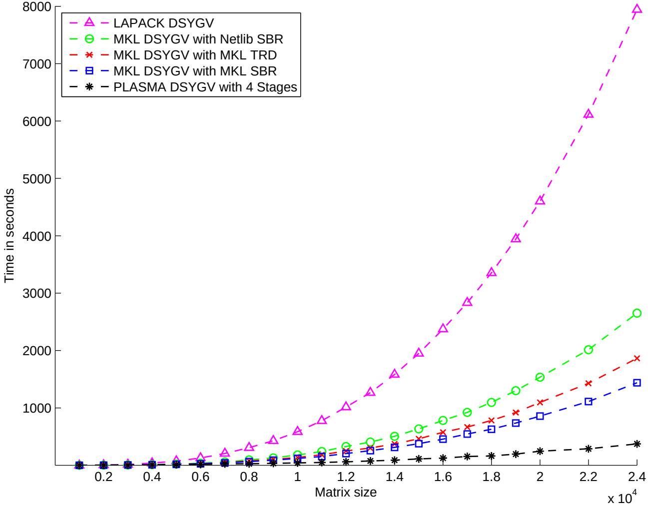
Slide courtesy from Azzam Haidar, ICL@UTK

PLASMA - Two-stage: BRD + Singular Value



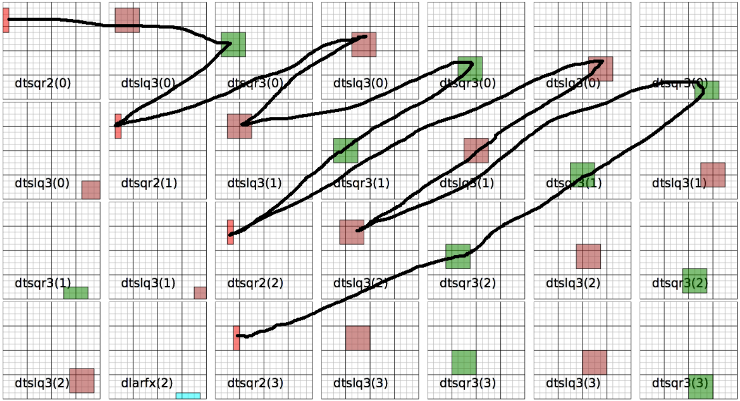
Azzam Haidar, Hatem Ltaief, Piotr Luszczek and Jack Dongarra, IPDPS'12

PLASMA - Four-stage: SYGV



Hatem Ltaief, Piotr Luszczek, Azzam Haidar and Jack Dongarra, ParCo'11

Bulge Chasing "ZigZag": Challenging!



Challenging because:

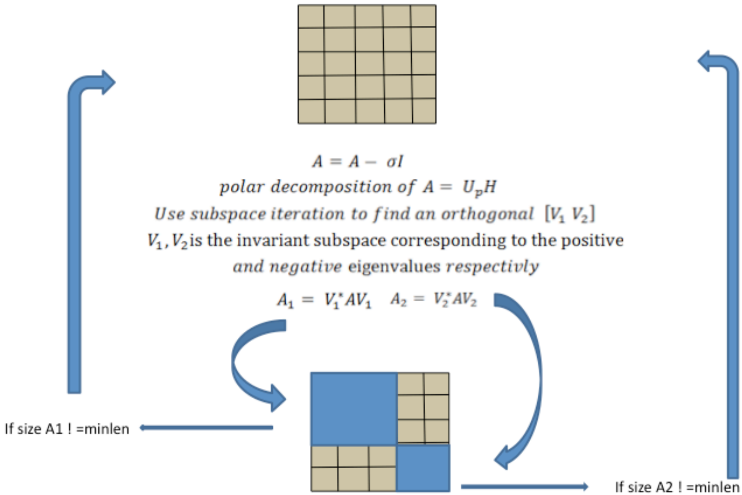
- Port on distributed memory systems
- Port on hardware accelerators
- Non-conventional computational kernels

Other Approach: QDWH

Y. Nakatsukasa and N. Higham, **Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD**, University of Manchester, MIMS EPrint: 2012.52, 2012.

- Spectral divide and conquer algorithm
- Can exploit fast, backward stable algorithms for computing the polar decomposition
- Uses just the computational kernels of Cholesky/QR factorizations (≤ 6 iterations) and GEMM

QDWH Procedure



Repeat the same procedure on resulting A1, A2 until we reach the minimum length

Accuracy/Orthogonality Formulas

For "BBQing" QDWH, we used the following formulas:

- To measure the accuracy of the computed eigenvalues compared to the reference eigenvalue δ, which are the exact eigenvalues (analytically known), or the ones computed by the QR iteration routine using LAPACK (DSTEV):

$$\frac{\|\lambda_i - \delta_i\|}{\|\lambda_i\|}$$

- To measure the orthogonality of the computed eigenvectors:

$$\frac{\|I - QQ^T\|}{\sqrt{n}}$$

Matrices with Known Eigenvalues

Type	Description
Type1	$\lambda_1 = 1, \lambda_i = \frac{1}{\mu lp}, fori = 2, ..., n$
Type2	$\lambda_i = 1, fori = 1, ..., n-1, \lambda_n = \frac{1}{\mu lp}$
Type3	$\lambda_i = \mu lp^{\frac{1-i}{n-1}}, fori = 1, ..., n$
Type4	$\lambda_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{k}), \lambda_i = \frac{1}{\mu lp}, fori = 2, ..., n$
Type5	n random numbers in $(1 - \frac{1}{k})$, their logarithms are uniformly distributed
Type6	n random numbers from a specified distribution
Type7	$\lambda_i = \mu lp \times i, fori = 1, ..., n-1, \lambda_n = 1$
Type8	$\lambda_1 = \mu lp, \lambda_i = 1 + \sqrt[2]{\mu lp} \times i, fori = 2, ..., n$
Type9	$\lambda_1 = 1, \lambda_i = \lambda_{i-1} + 100 \times \mu lp, fori = 2, ..., n$

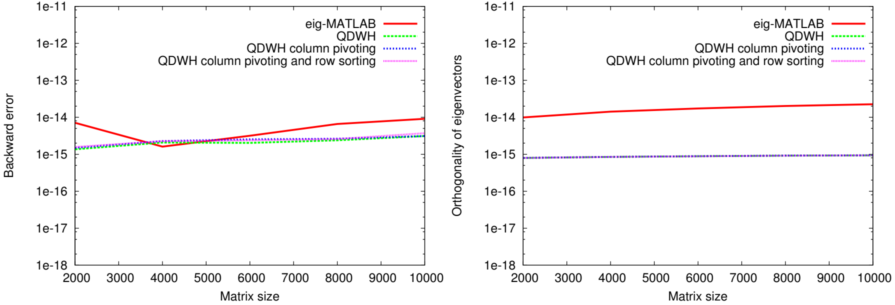
Matrices with Interesting Properties

Type	Description
Type10	Tridiagonal Matrix
Type11	Wilkinson Tridiagonal Matrix
Type12	Clement Tridiagonal Matrix
Type13	Legendre Tridiagonal Matrix
Type14	Laguerre Tridiagonal Matrix
Type15	Hermite Tridiagonal Matrix

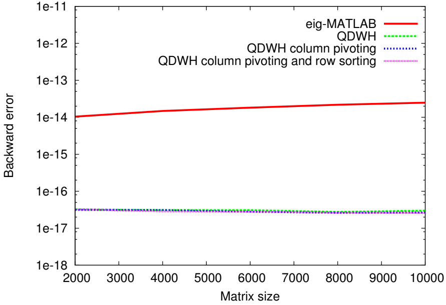
Matrices from Real Life Applications

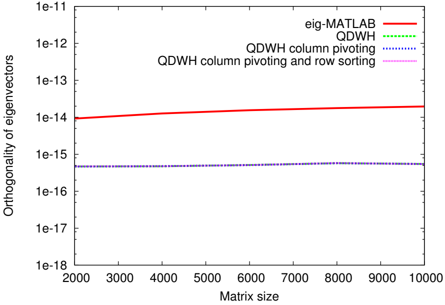
Type	Description
Type16	Matrices from application on Quantum Chemistry and Electronic structure
Type17	The bcsstruc1 set in Harwell-Boeing collection
Type18	Matrices from Almedar, Nasa, and Cannizzo Sets in the University of Florida

Matrix Type 1

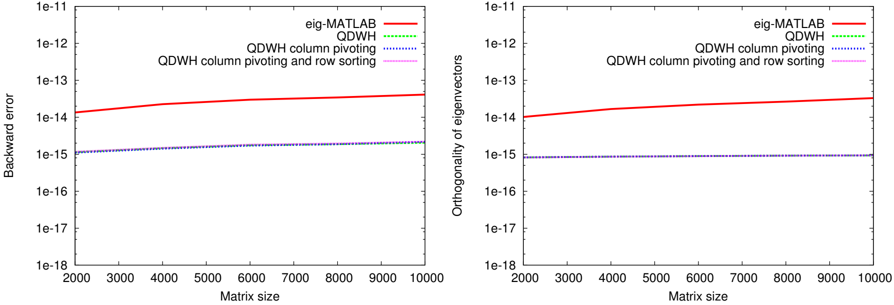


Matrix Type 2

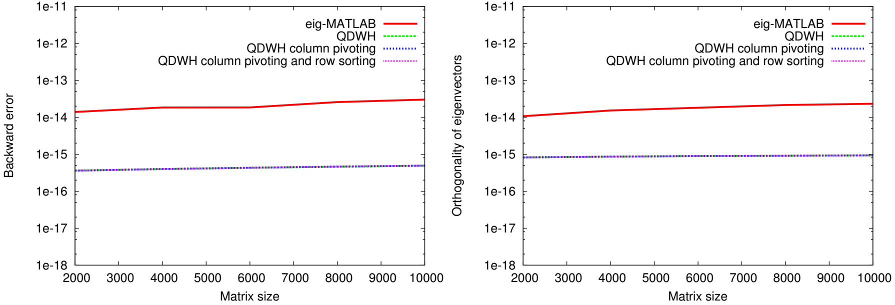




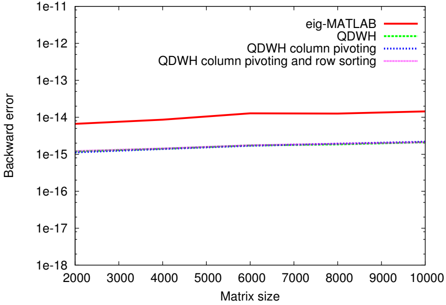
Matrix Type 3

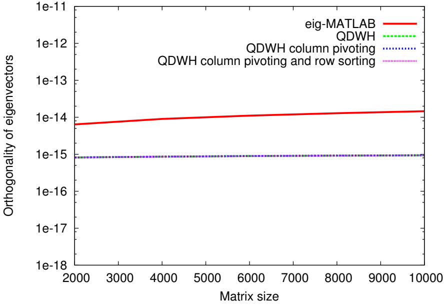


Matrix Type 4

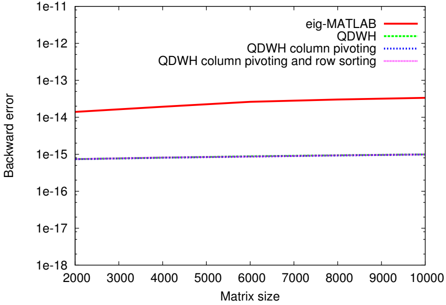


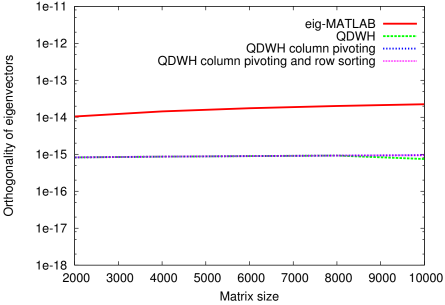
Matrix Type 5



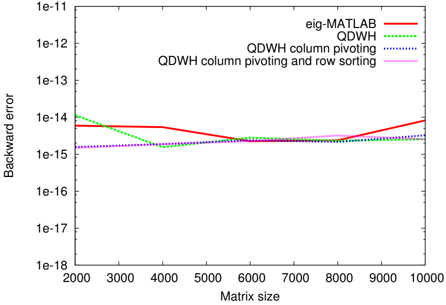


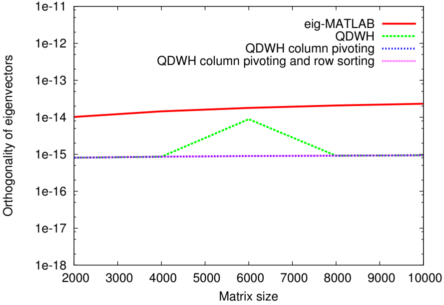
Matrix Type 6



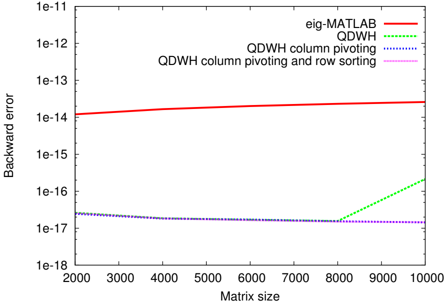


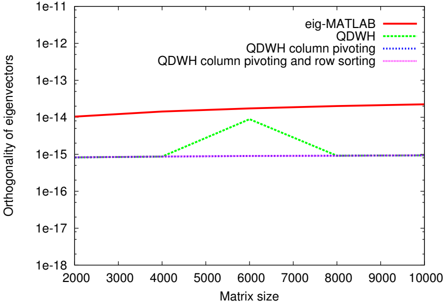
Matrix Type 7



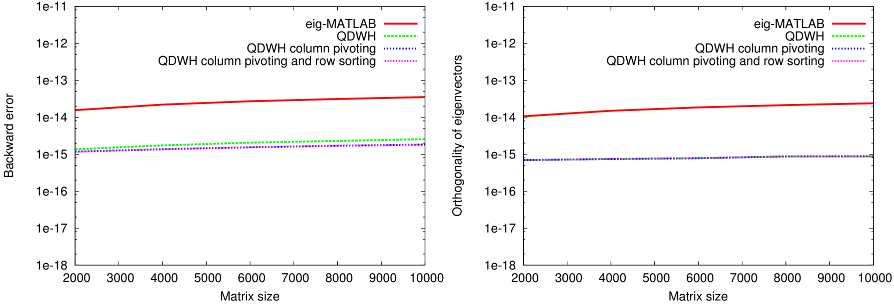


Matrix Type 8

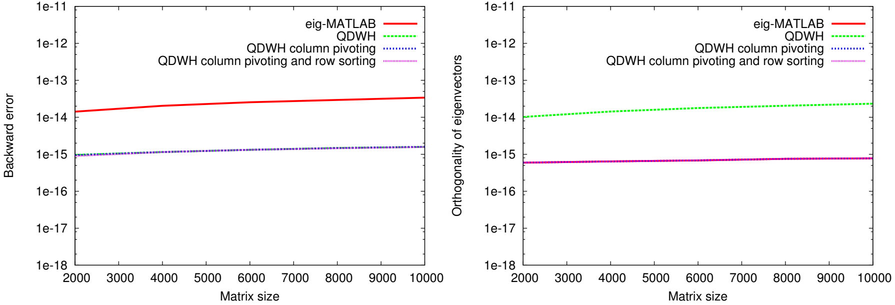




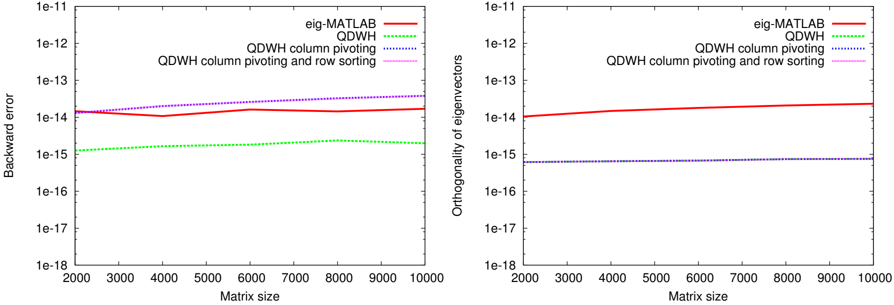
Matrix Type 12 Clement



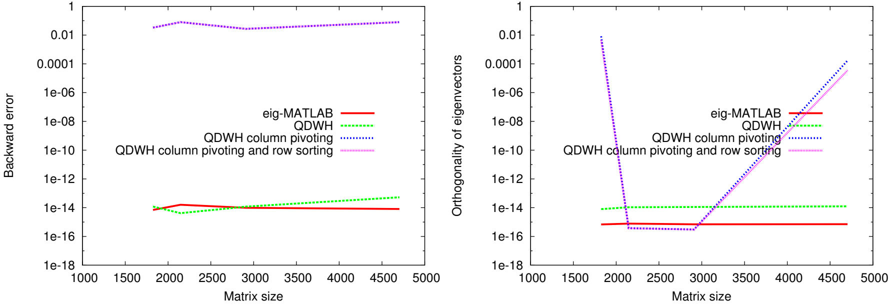
Matrix Type 14 Laguerre



Matrix Type 15 Hermite



Matrix Type 18 NASA



Environment Settings

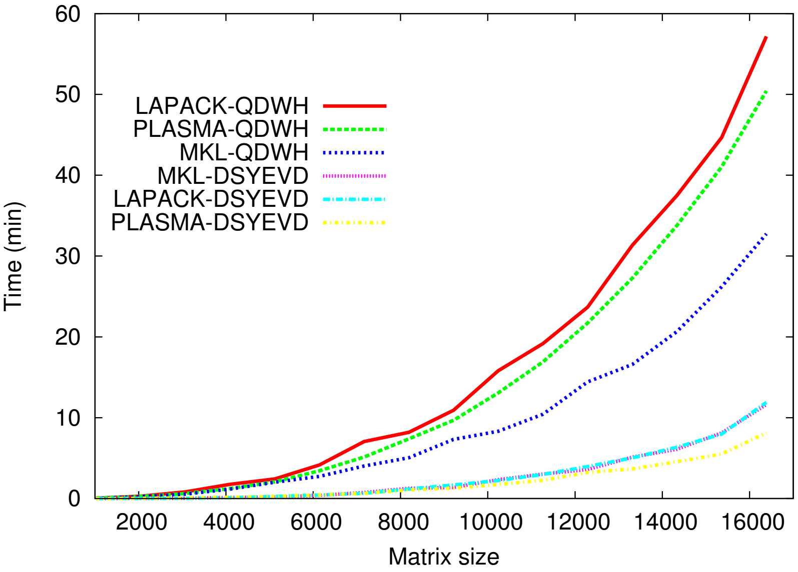
Hardware:

- Intel(R) Xeon(R) CPU E5-2650
- Dual-socket 8-core (16 cores total) and up to 32 threads with hyper-threading
- 20M Cache, 2.00 GHz, 8.00 GT/s Intel(R) QPI
- AVX Instruction Set
- 65GB of DDR3 main memory

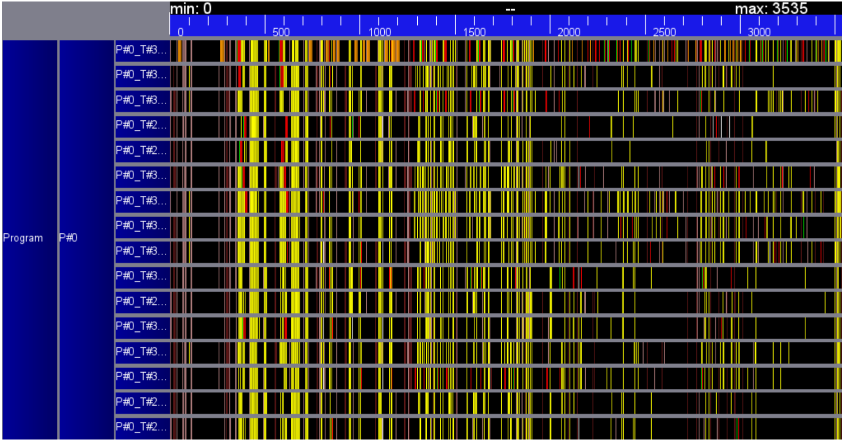
Software:

- Intel Compiler Suite 2013.1.117
- PLASMA 2.4.5 (compiled w/ -mkl=sequential)
- LAPACK 3.4.2 (compiled w/ -mkl=parallel)

Performance Comparisons - Matrix Type 1

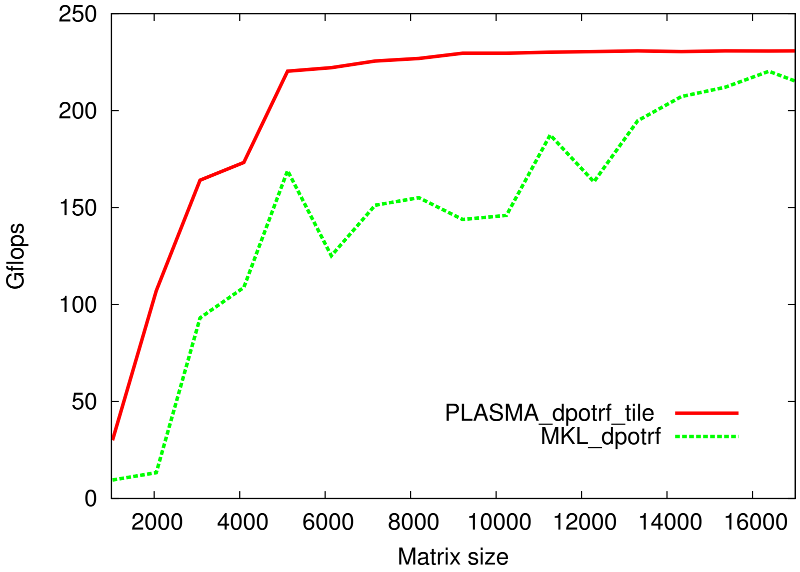


PLASMA-QDWH - QUARK/ViTE Trace

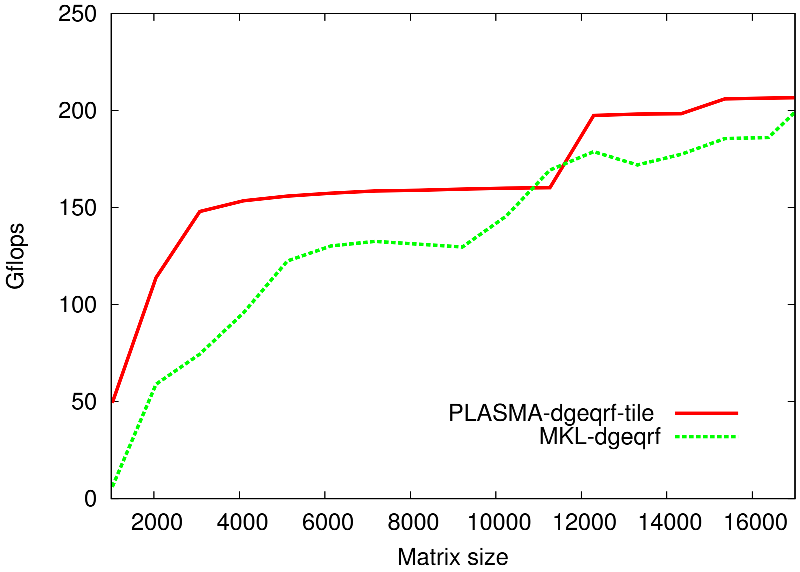


Naive PLASMA-QDWH implementation!

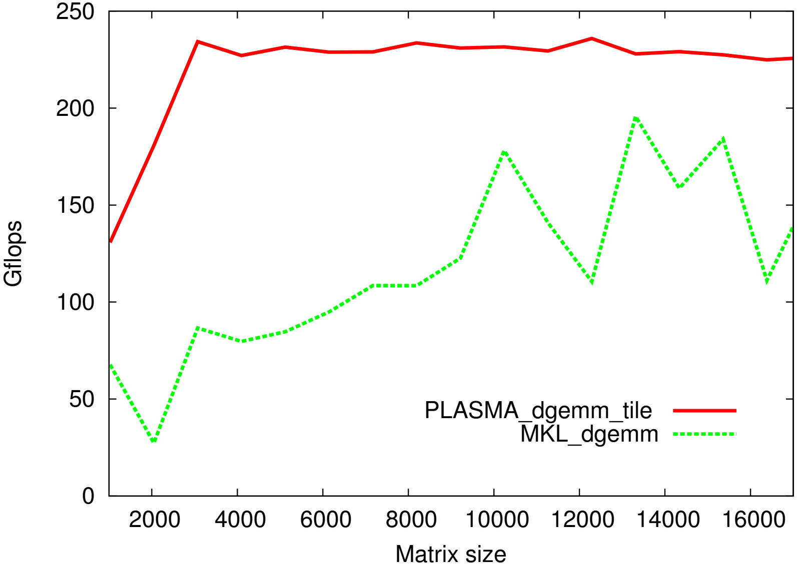
Cholesky Factorization



QR Factorization

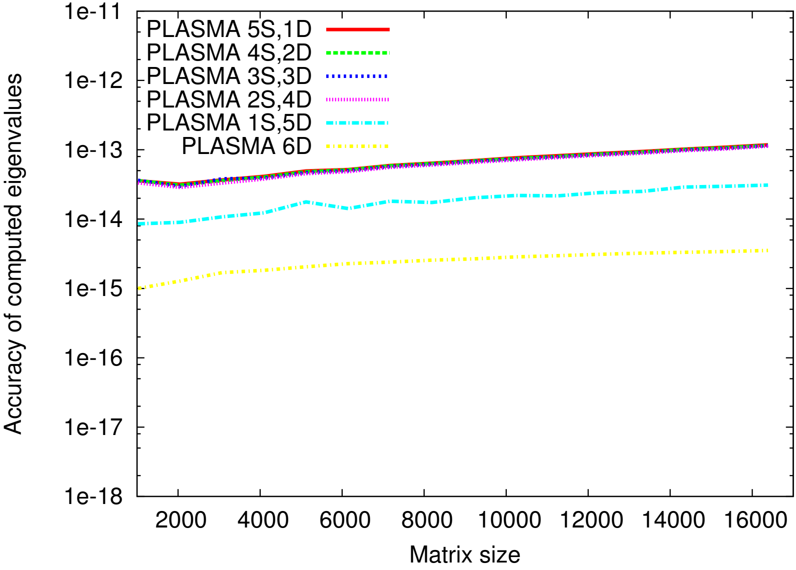


DGEMM

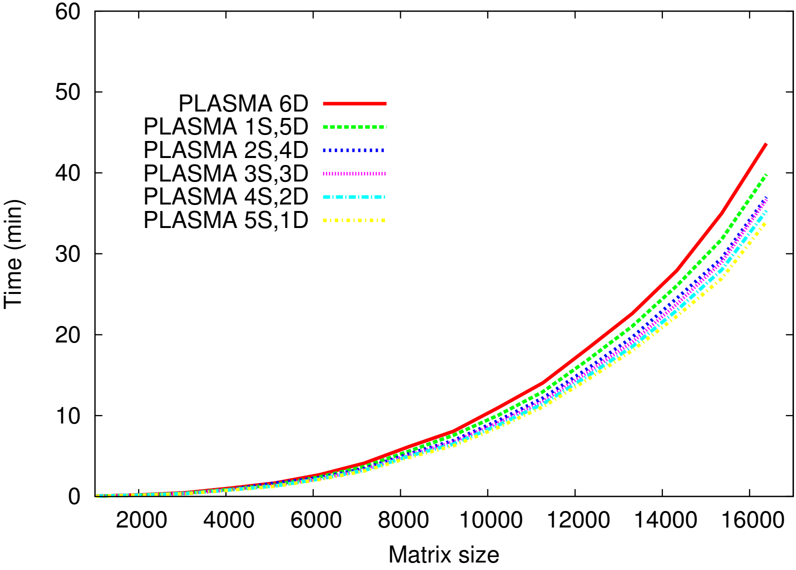


Gflops
0
50
100
150
200
250
2000
4000
6000
8000
10000
12000
14000
16000
Matrix size
PLASMA_dgemm_tile
MKL_dgemm

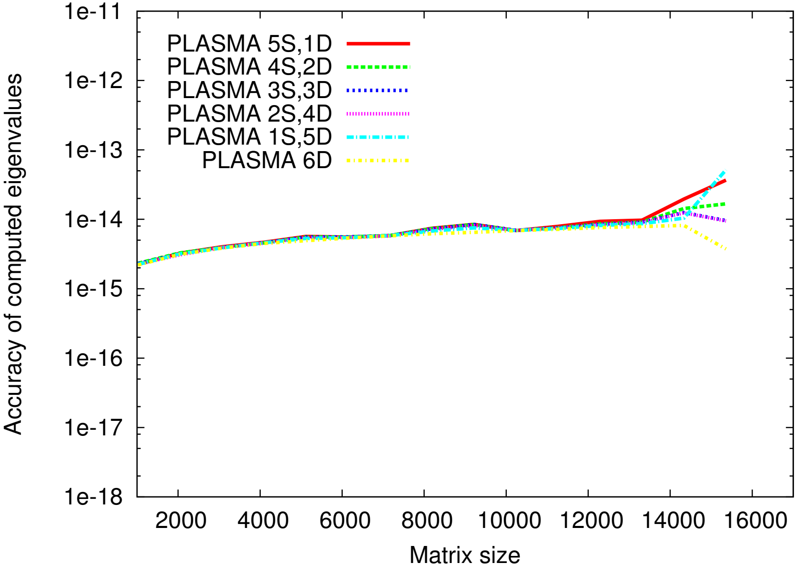
Mixed Precisions - Accuracy - Matrix Type 1



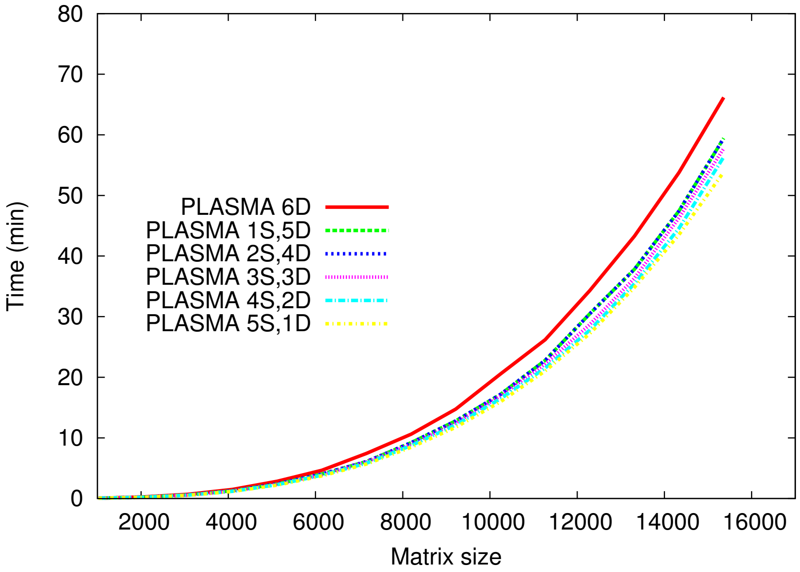
Mixed Precisions - Performance - Matrix Type 1



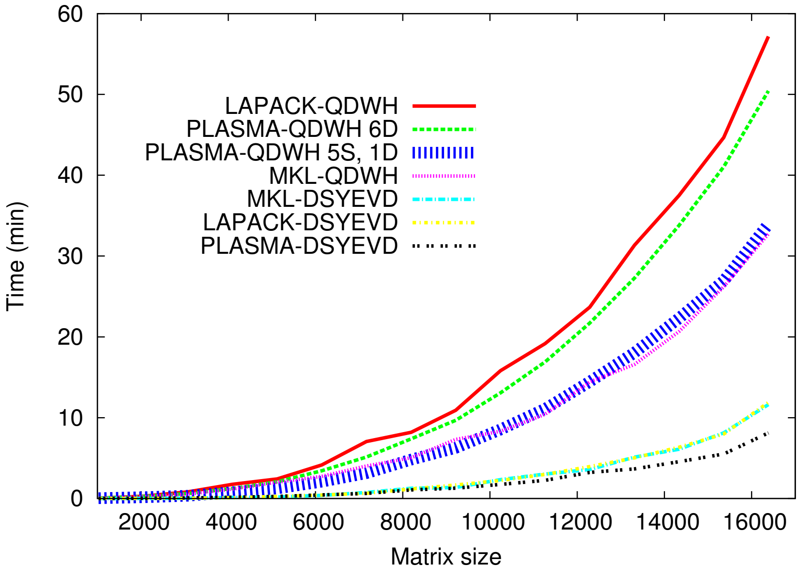
Mixed Precisions - Accuracy - Matrix Type 2



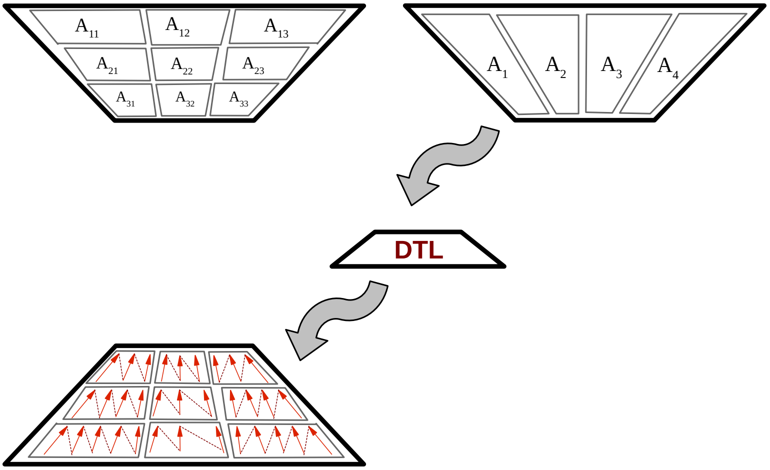
Mixed Precisions - Performance - Matrix Type 2



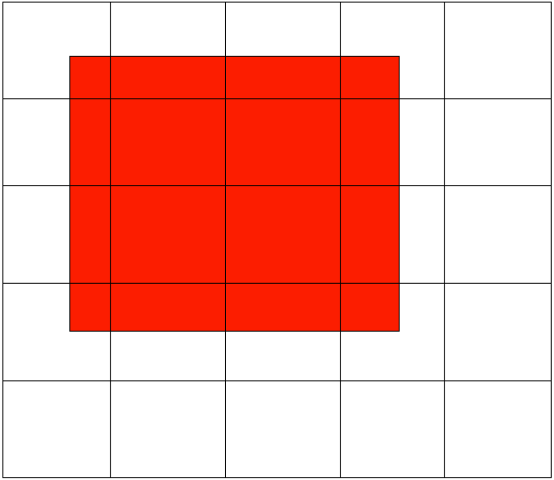
Overall Performance - Matrix Type 1



Data Translation Layer: Matching Algorithms with Storage



Data Translation Layer - Productivity



How about adjusting the shift?

What is Next for PLASMA-QDWH?

- Currently Mixed Precision PLASMA-QDWH as good as MKL-QDWH due to the overhead of data translation back and forth: very promising!.
- DTL will help in removing these data format translations.
- "_Tile" interface (advanced mode).
- "Tile_Async" interface (expert mode).
- Fine-grain computations to remove unnecessary computations: for instance, running QR of a dense matrix on top of an Id at each iteration w/o taking into account the structure of the global matrix.
- Reduce first the matrix to band form and run QDWH on it: however, band structure not conserved throughout the iterations.
- On distributed memory systems, highly optimized Hierarchical QR (UCD) and 2.5D GEMM (UCB) should help QDWH flying.
- Eigenvectors...