Divide and Conquer Symmetric Tridiagonal Eigensolver for Multicore Architectures

29th IEEE International Parallel & Distributed Processing Symposium, Hyderabad 2015

Grégoire Pichon, Azzam Haidar, Mathieu Faverge, Jakub Kurzak

May 26, 2015

Scientific Context

PLASMA

- Dense linear algebra library
- Designed for shared-memory systems

Symmetric Eigenproblem

- Solve $AX = \Lambda X$, where A is a symmetric square matrix
- Λ are the eigenvalues, X the eigenvectors

Eigenproblems in PLASMA

Three Stages Symmetric Eigensolvers

1. $A = QTQ^T$: reduction to tridiagonal form: $\Theta(n^3)$
2. $T = V\Lambda V^T$: tridiagonal eigensolver
3. $A = (QV)\Lambda(QV)^T$: back-transformation: $\Theta(n^3)$

Q, V orthogonal; T tridiagonal

Objectives

- Introduce a task-based tridiagonal eigensolver
- Towards a fully task-based eigensolver in PLASMA

Outline

Tridiagonal Eigenproblems in LAPACK

Algorithm	Operational Cost	Workspace	Accuracy
QR	$\Theta(n^3)$	$\Theta(n)$	$\Theta(\sqrt{n}\epsilon)$
BI	$\Theta(n^3)$	$\Theta(n)$	$\Theta(n\epsilon)$
DC	$\Theta(n^2)$ to $\Theta(n^3)$	$\Theta(n^2)$	$\Theta(\sqrt{n}\epsilon)$
MRRR	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n\epsilon)$

Computing all eigenpairs

Particular Cases

- Computing all eigenvalues: $\Theta(n^2)$ using QR-PWK
- Computing k eigenpairs: $\Theta(nk)$ using MRRR

Computing All Eigenpairs

MRRR vs D&C

- Depend on the matrix properties: “clustered” eigenvalues
- The average performance on practical matrices for a sequential run is comparable (*Performance and Accuracy of LAPACK’s Symmetric Tridiagonal Eigensolvers* by Demmel et al.)
  - D&C: $\Theta(n^{2.5})$
  - MRRR: $\Theta(n^{2.2})$

Choice

- D&C was not implemented in a task-based fashion
- Depend on the application: Cost vs Accuracy

Outline

Divide & Conquer – Divide

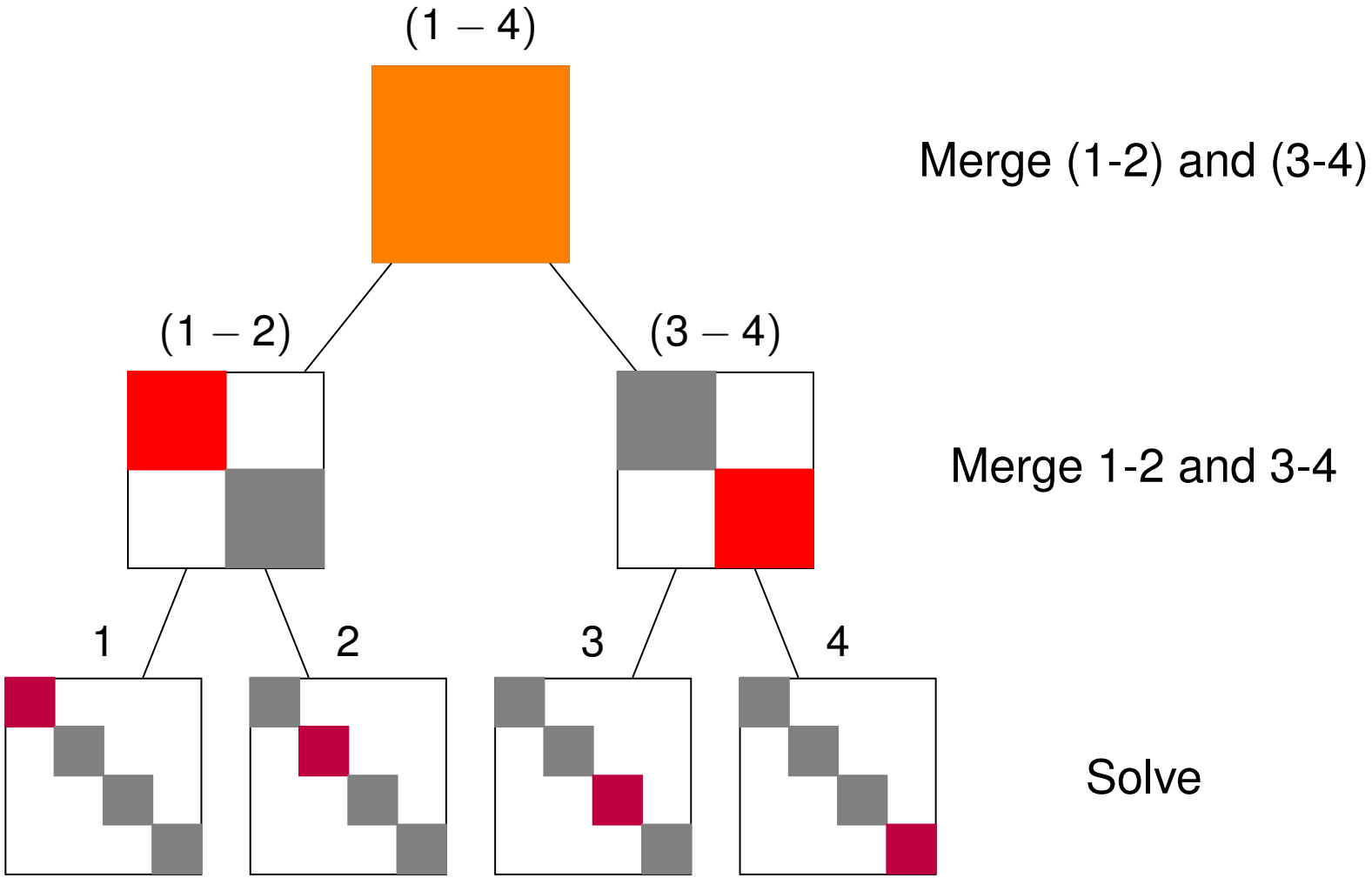
Strategy

1. Divide the problem into two halves subproblems
2. Recursively solve each subproblem
3. Merge those subproblems according to the approximation

Divide with a rank-1 approximation:

$$\begin{pmatrix} * & * & \emptyset & \emptyset \\ * & * & \beta & \emptyset \\ \emptyset & \beta & * & * \\ \emptyset & \emptyset & * & * \end{pmatrix} = \begin{pmatrix} * & * & \emptyset & \emptyset \\ * & *' & \emptyset & \emptyset \\ \emptyset & \emptyset & *' & * \\ \emptyset & \emptyset & * & * \end{pmatrix} + \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \beta & \beta & \emptyset \\ \emptyset & \beta & \beta & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

Divide & Conquer – Conquer



Mathematical Background

Since T has been decomposed as:

$$T = \begin{pmatrix} V_1 D_1 V_1^T & \emptyset \\ \emptyset & V_2 D_2 V_2^T \end{pmatrix} + \beta u u^T$$

We have:

$$T = \tilde{V}(D + \beta z z^T)\tilde{V}^T$$

The merging equation is:

$$(D + \beta z z^T) = X \Lambda X^T$$

Final T eigenvectors are:

$$V = \tilde{V} \times X$$

Relations

- $D = diag(D_1, D_2)$
- $\tilde{V} = diag(V_1, V_2)$
- $z = \tilde{V}^T u$

Deflation Process

The deflation occurs in $(D + \beta zz^T)$ when

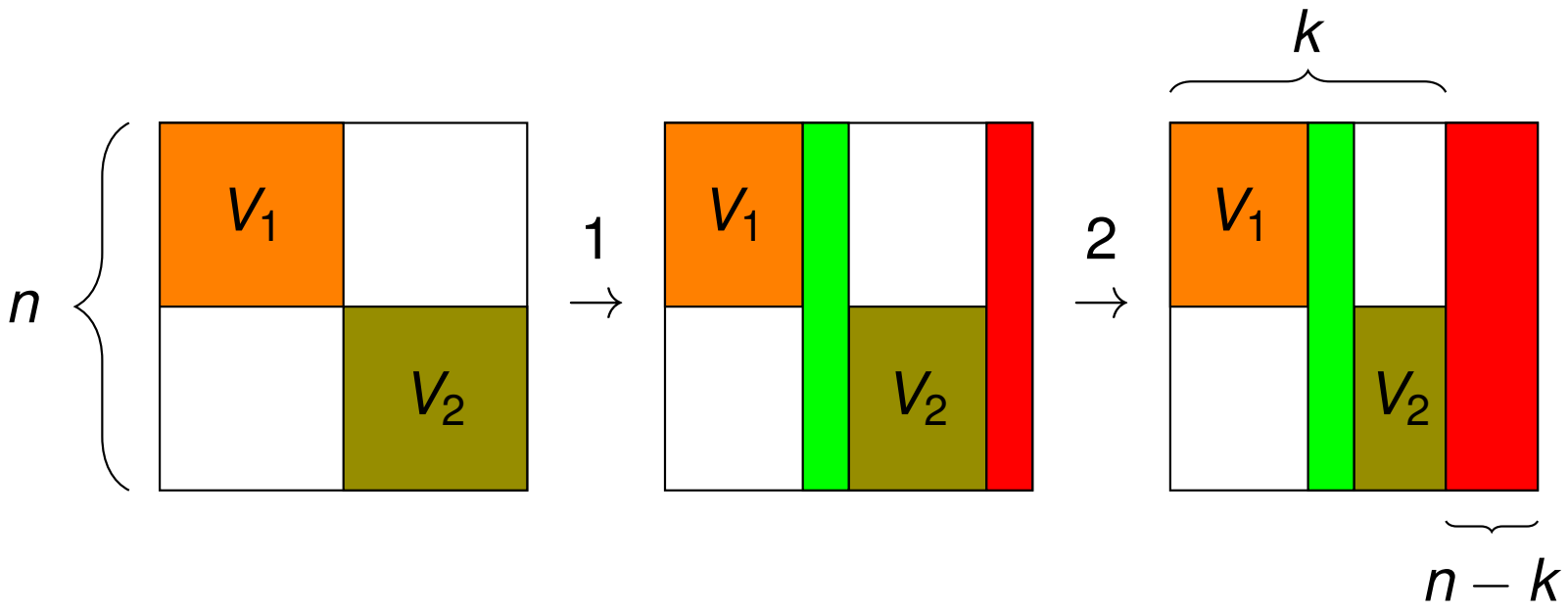
- An eigenvalue has a multiplicity greater than one
- An entry in z is null

For those eigenvalues, the solution provided by the subproblems is also the solution of the current problem

Merging Phase

1. Find the deflated eigenvalues
2. Solve the merging equation for non-deflated eigenvalues
3. Update the previous system for non-deflated eigenvalues

Merge Step - Deflate

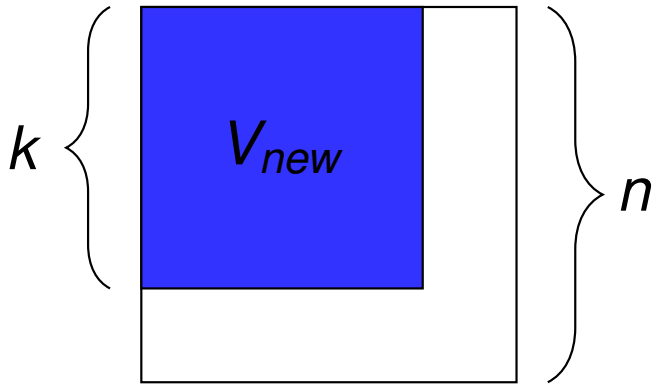


1: Deflation due to equal eigenvalues from D_1 and D_2
2: Deflation due to zero entries in z

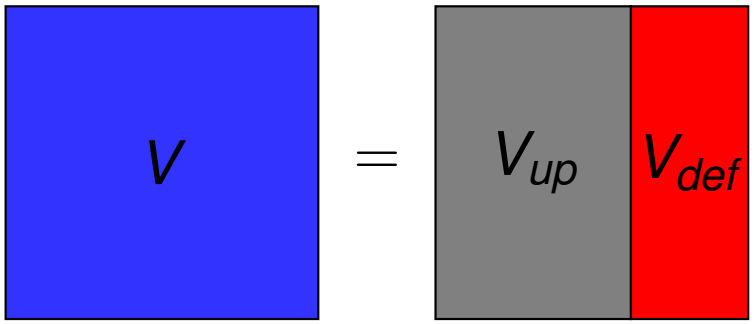
Merge Step - Compute Eigenvectors

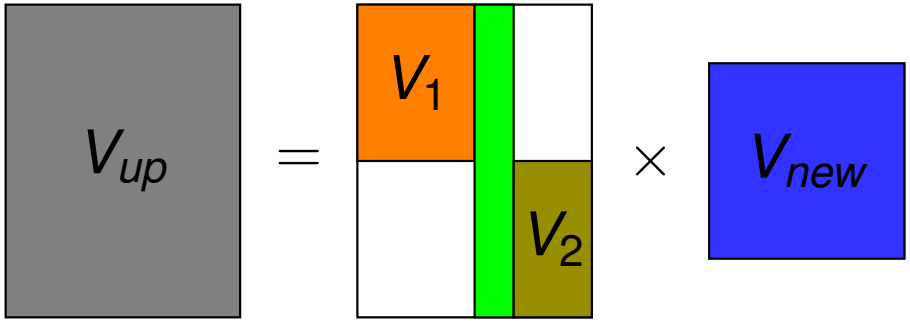
Operations on Non-Deflated Eigenvectors

1. Solve the secular equation (rely on LAPACK)
2. Reduction to compute the stabilization vector W
3. Stabilize eigenvectors according to $W \rightarrow V_{new}$



Merge Step - Update Eigenvectors





Performance

- V_{up}: computational bound
- V_{def}: memory bound

Outline

Related Work

MKL LAPACK

- LAPACK algorithm linked with optimised BLAS 3 kernels
- Only GEMMs in parallel
- Interesting speedup: GEMMs represent 80% of the overall time

MKL ScaLAPACK

- Independent subproblems can be solved in parallel
- GEMMs in parallel, as well as solving secular equations
- Permutations are different with LAPACK version
- Main limit: static scheduling

PLASMA Programming Model

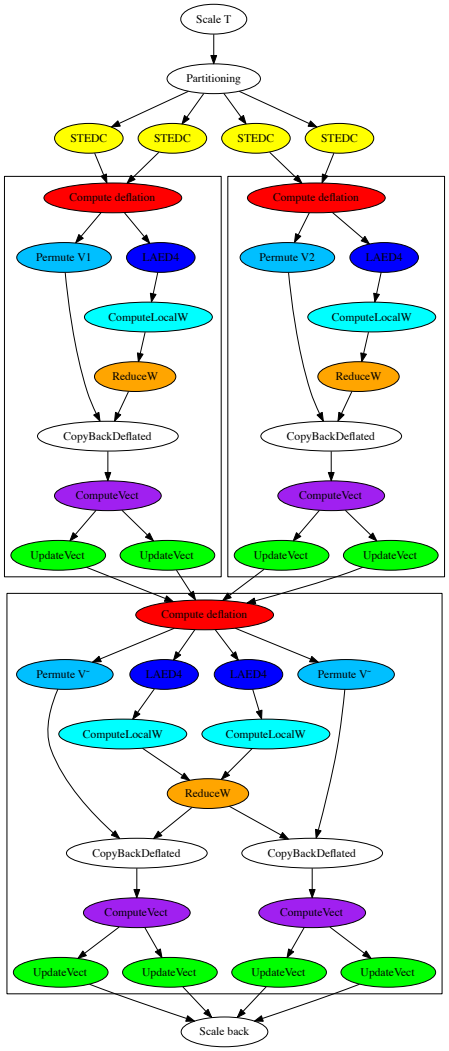
Using a Direct Acyclic Graph (DAG)

- Algorithm expressed as a flow of sequential tasks
- Rely on the QUARK dynamic runtime
- Operations on vectors panels (1D)

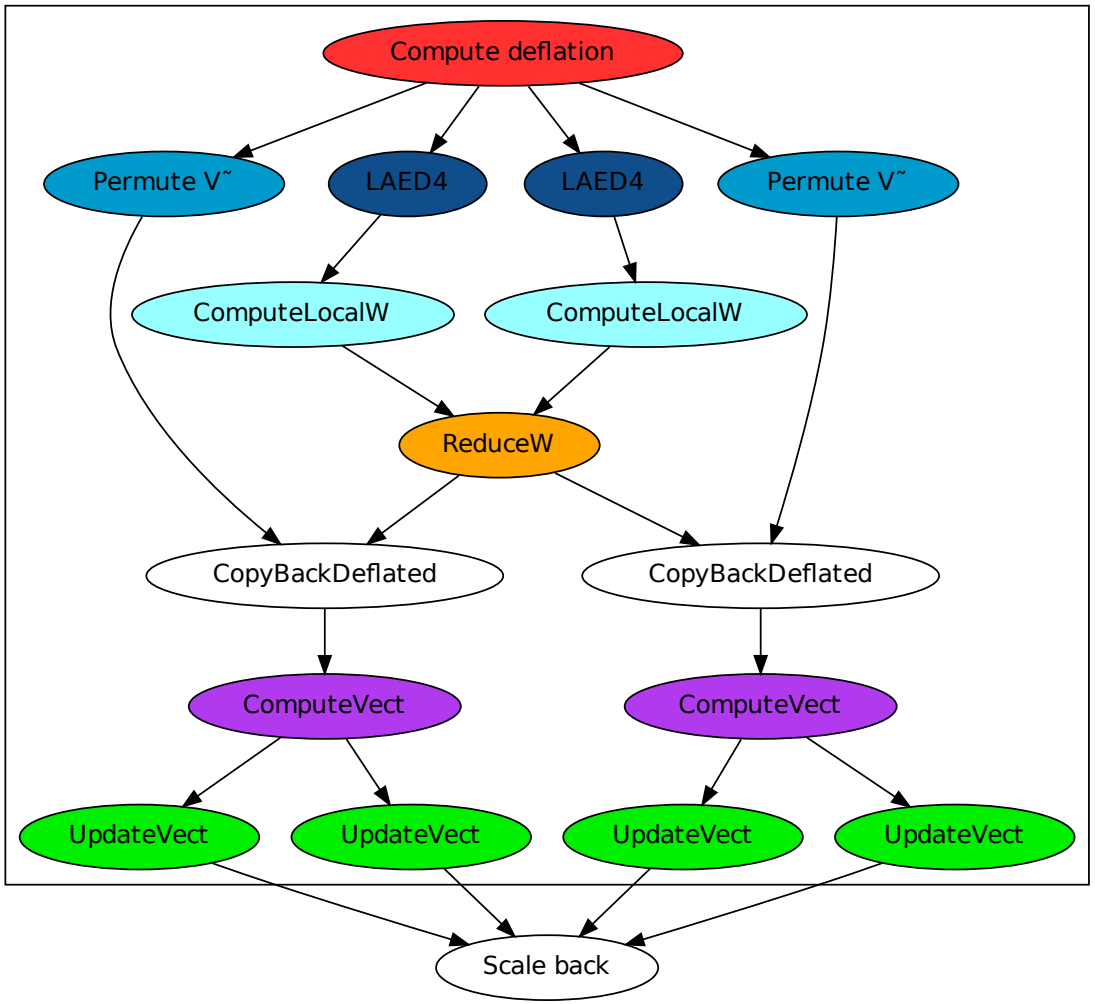
Dependencies

- INPUT: read
- INOUT: read/write
- OUTPUT: write

Direct Acyclic Graph (1)



Direct Acyclic Graph (2)

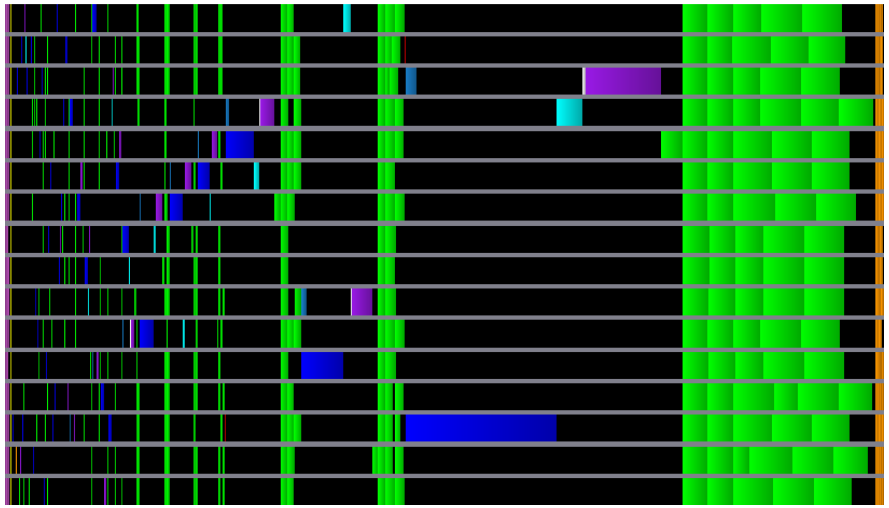


Properties

- Generic DAG
- Constant number of dependencies

Without Deflation - Parallel GEMMs

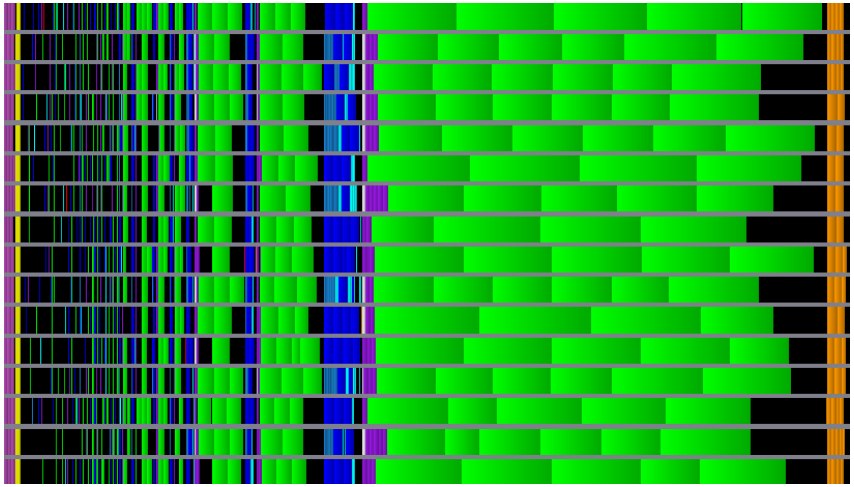
Traces on a n-by-n matrix of type 4 with $n = 10\ 000$ on a 16 cores machine



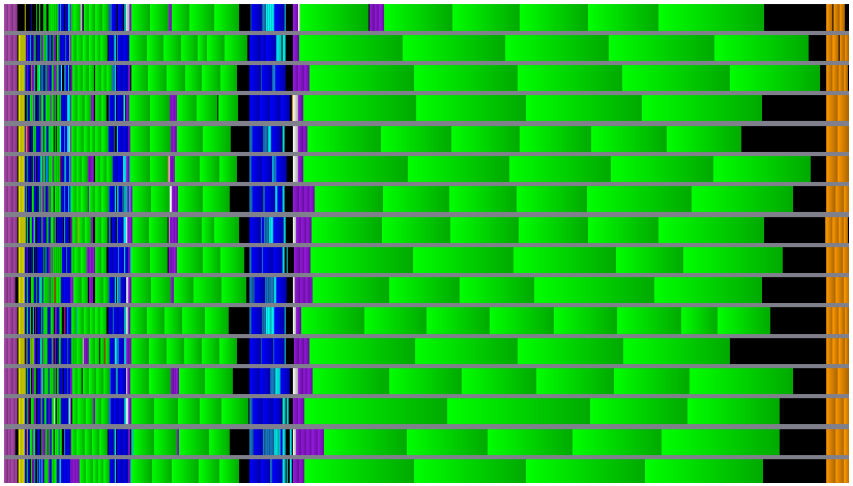
Equivalent to MKL LAPACK (4.3s)

	UpdateVect		*ComputeVect*
	LAED4		*ComputeLocalW*

Without Deflation - Parallel Merge

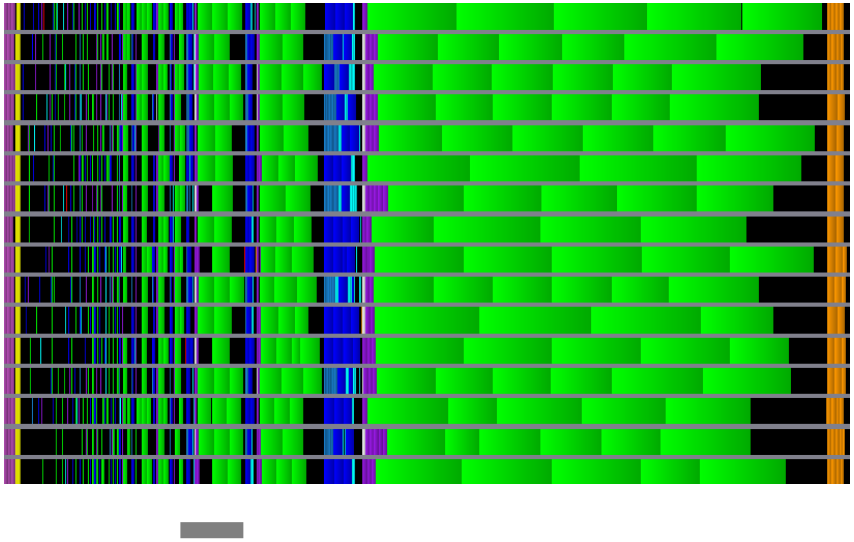


Operations within a merge in parallel (1.8s)

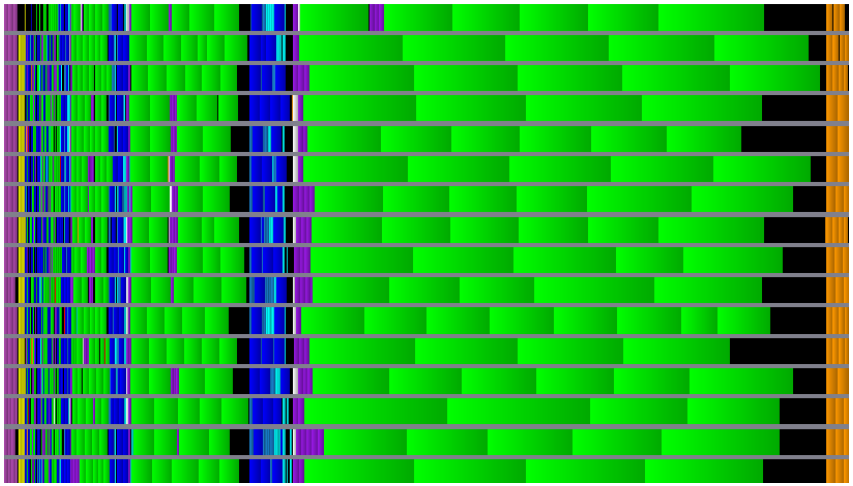


Independent subproblems in parallel (1.6s)

Without Deflation - Parallel Merge

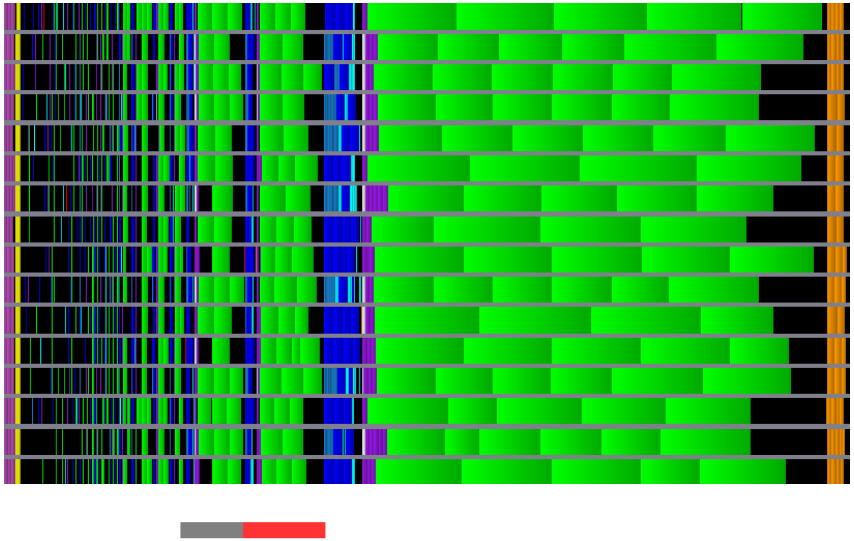


Operations within a merge in parallel (1.8s)

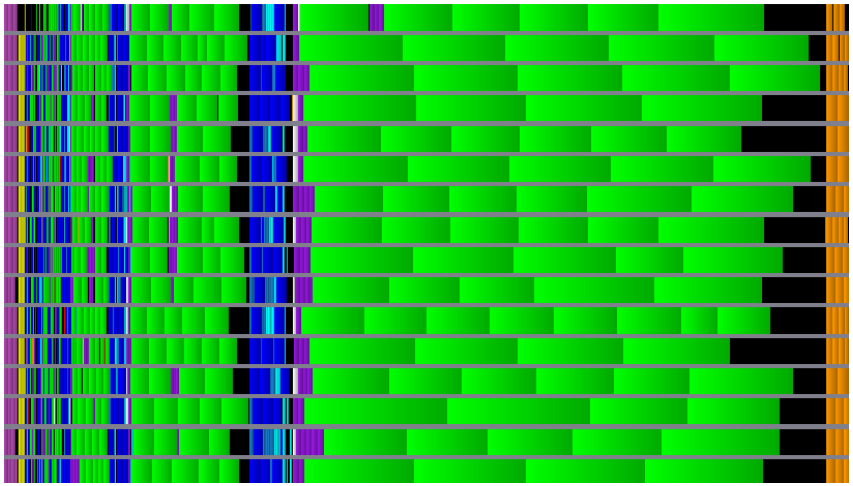


Independent subproblems in parallel (1.6s)

Without Deflation - Parallel Merge

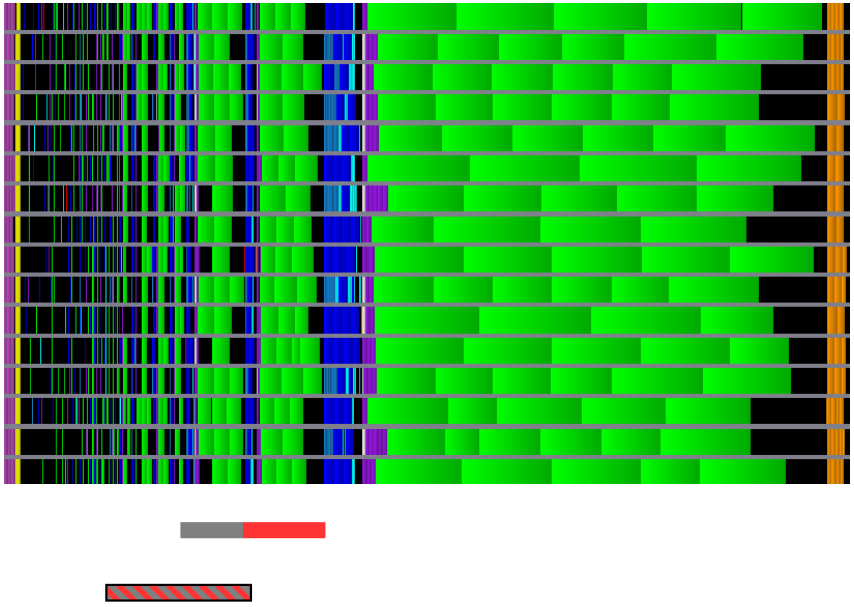


Operations within a merge in parallel (1.8s)

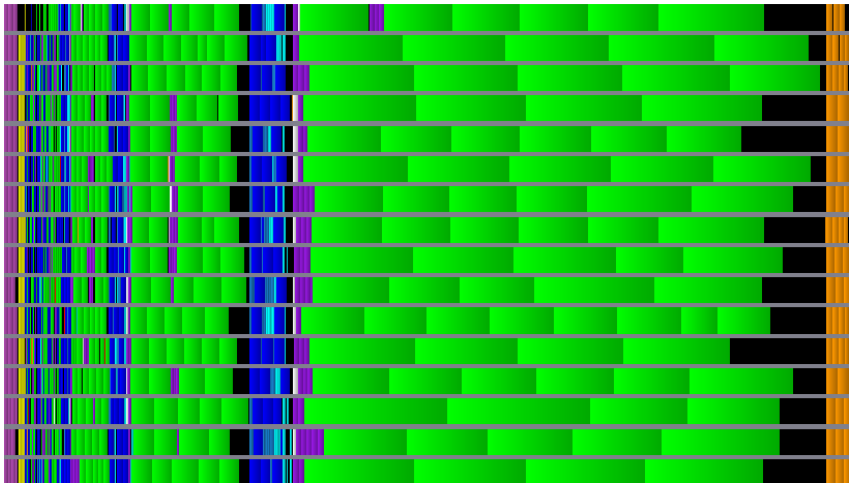


Independent subproblems in parallel (1.6s)

Without Deflation - Parallel Merge



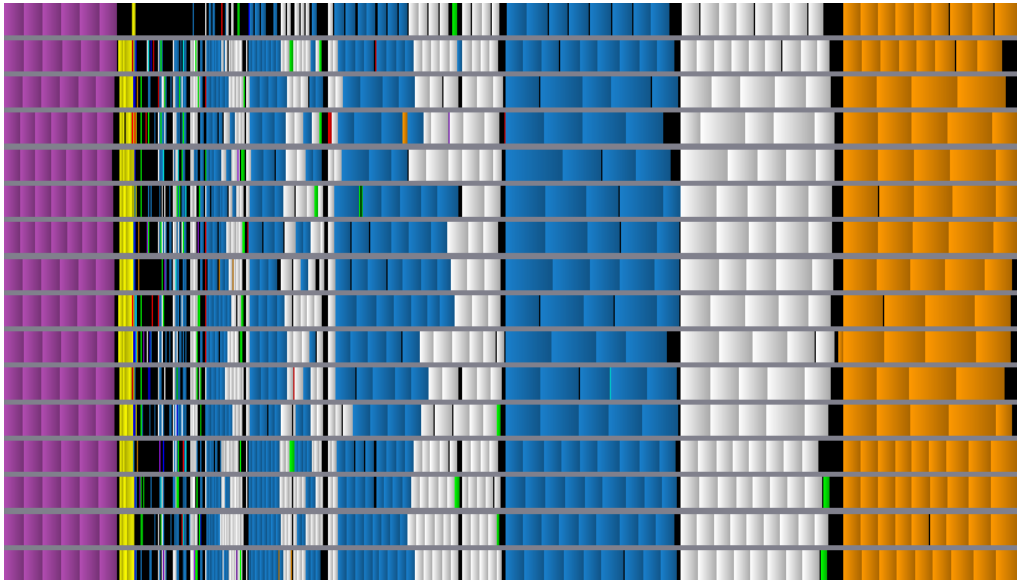
Operations within a merge in parallel (1.8s)



Independent subproblems in parallel (1.6s)

With Deflation

Trace on a n-by-n matrix of type 2 with $n = 10\ 000$
on a 16 cores machine (0.4s)



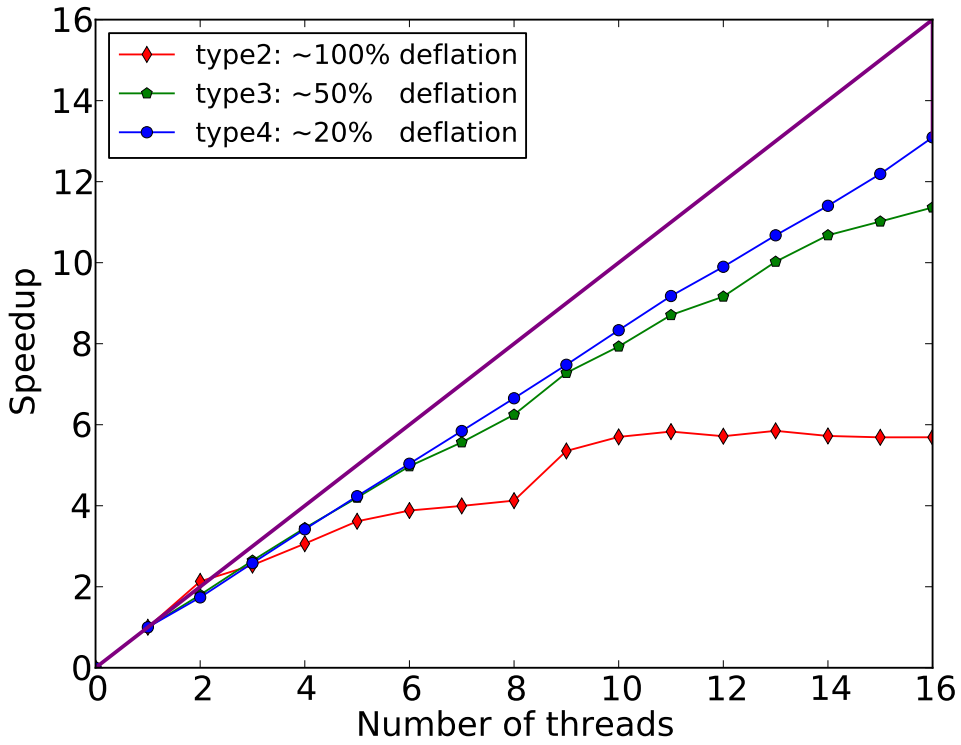
	Permute $\bar{V}$		*CopyBackDeflated*
	LASET		*SortEigenvectors*

Outline

Validation on a Large Set of Matrices

Type	Description
1	$\lambda_1 = 1$, $\lambda_{i,i\in[2,n]} = \frac{1}{k}$
2	$\lambda_{i,i\in[1,n-1]} = 1$, $\lambda_n = \frac{1}{k}$
3	$\lambda_{i,i\in[1,n]} = k^{-\frac{i-1}{n-1}}$
4	$\lambda_{i,i\in[1,n]} = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{k})$
5	n random numbers with $log(i)$ uniformly distributed
6	n random numbers
7	$\lambda_{i,i\in[1,n-1]} = ulp \times i$, $\lambda_n = 1$
8	$\lambda_1 = ulp$, $\lambda_{i,i\in[2,n-1]} = 1 + i \times \sqrt{ulp}$, $\lambda_n = 2$
9	$\lambda_1 = 1$, $\lambda_{i,i\in[2,n]} = \lambda_{i-1} + 100 \times ulp$
10	(1,2,1) tridiagonal matrix
11	Wilkinson matrix
12	Clement matrix
13	Legendre matrix
14	Laguerre matrix
15	Hermite matrix

Scalability with 16 cores and $n = 10\ 000$



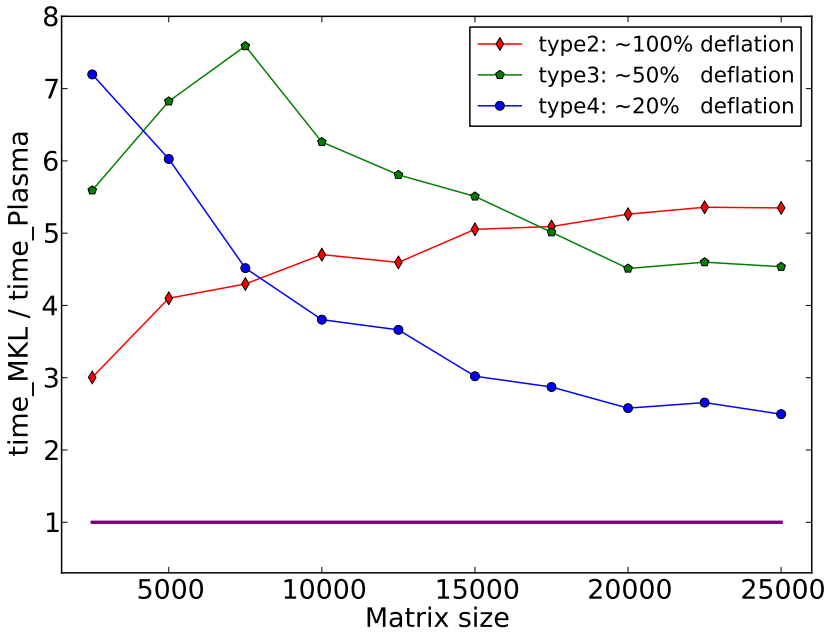
Different levels of deflation

Machine

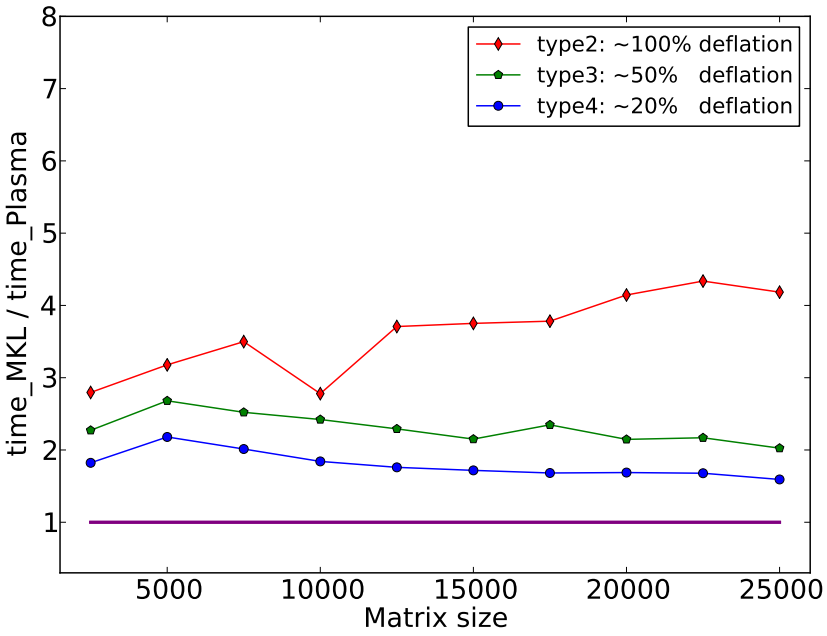
- Two INTEL XEON E5-2650v2 at 2.6GHz
- 64 GB of memory

Relative speedup vs Existing Solutions

With 16 cores



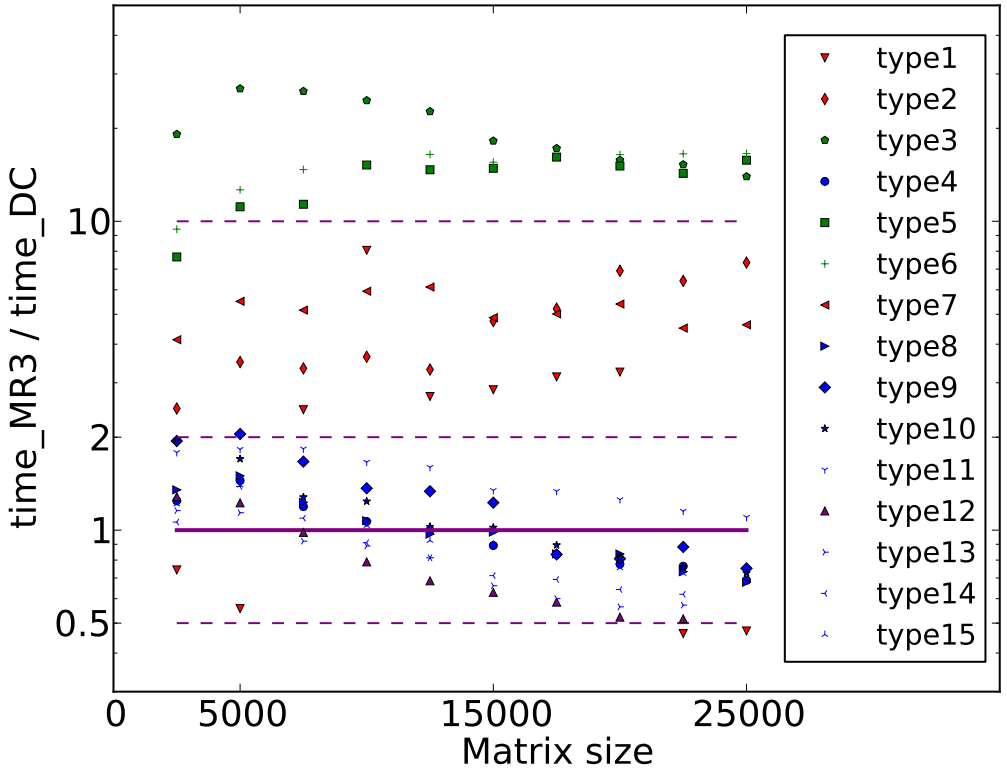
MKL LAPACK



MKL ScaLAPACK

Relative speedup vs MR3-SMP

With 16 cores

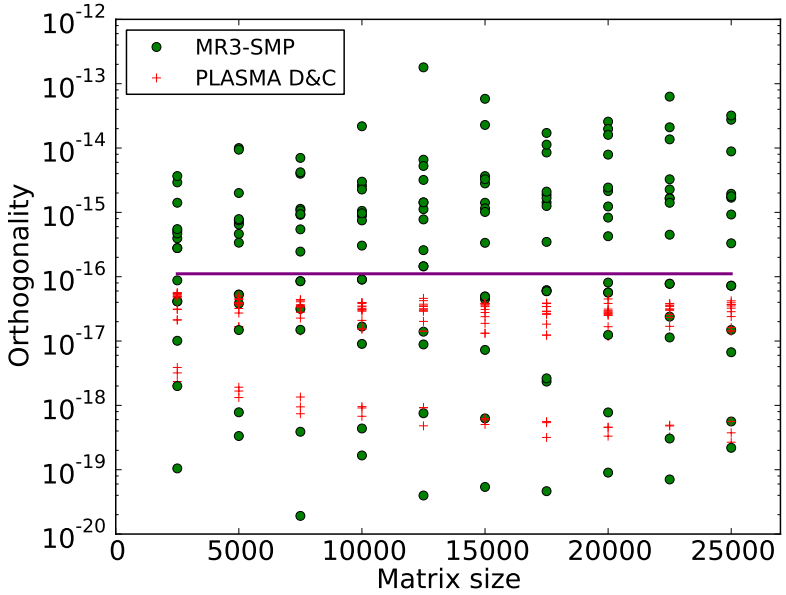


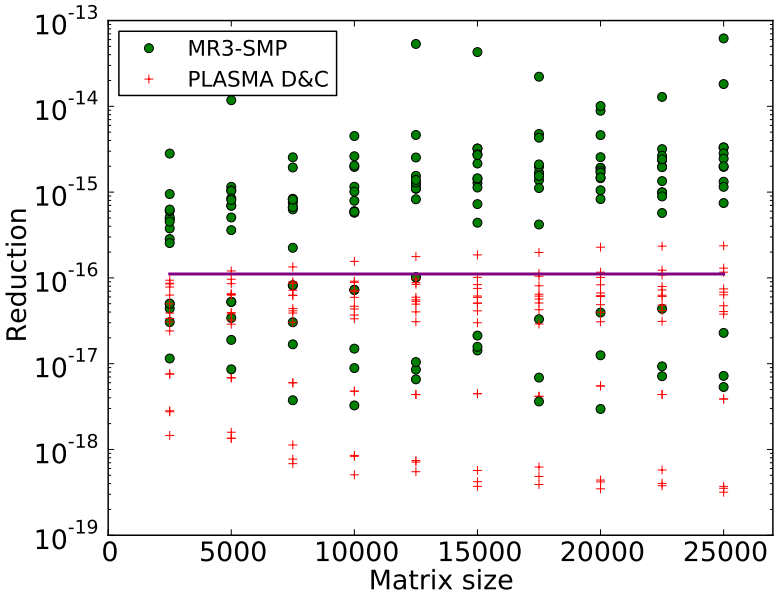
MR^3-SMP

- FLAME project
- Task-based approach for multicore architectures

Accuracy vs MRRR

Stands for classical LAPACK implementations





Orthogonality: $\frac{||Id - VV^T||}{n}$

Reduction: $\frac{||T - V\Lambda V^T||}{||T|| \times n}$

Conclusion

Results

- Outperforms existing D&C implementations
- Competitive with respect to MR^3-SMP
- Demonstrates the asset of a task-based approach

Future Work

- Solve the SVD problem using the same approach
- Expand to distributed/heterogeneous machines
- Gather with the reduction to tridiagonal form

Thank you!

Available in PLASMA 2.7.1

http://icl.cs.utk.edu/plasma/