Recent Advances in Dense Matrix Computations for Two-Sided Reduction Algorithms

Azzam Haidar[1] **Hatem Ltaief**[2] Piotr Łuszczek[1]
Jack Dongarra[1]

[1]Innovative Computing Laboratory

[2]KAUST Supercomputing Laboratory

London, UK, Jun 28-30, 2012

Outline

Matrices Over Runtime Systems at Exascale

$\Longrightarrow$ Mission statement: *"Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems"*.

$\Longrightarrow$ Separation of concerns:

- Algorithmic challenges to exploit the hardware capabilities at most.
- Runtime challenges due to the ever growing hardware complexity.

$\Longrightarrow$ MORSE Algorithms: MORSE-Dense (PLASMA, MAGMA, tile algorithms), MORSE-Sparse (PaStiX, MaPHyS), MORSE-Stencil, MORSE-FMM

$\Longrightarrow$ MORSE Runtimes: QUARK, StarPU, DAGuE

Matrices Over Runtime Systems at Exascale

$\Longrightarrow$ Mission statement: *"Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems"*.

$\Longrightarrow$ Separation of concerns:

- Algorithmic challenges to exploit the hardware capabilities at most.
- Runtime challenges due to the ever growing hardware complexity.

$\Longrightarrow$ MORSE Algorithms: MORSE-Dense (PLASMA, MAGMA, tile algorithms), **MORSE-Sparse (PaStiX, MaPHyS)**, MORSE-Stencil, MORSE-FMM

$\Longrightarrow$ MORSE Runtimes: QUARK, StarPU, **DAGuE**

Matrices Over Runtime Systems at Exascale

$\Longrightarrow$ Mission statement: *"Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems"*.

$\Longrightarrow$ Separation of concerns:

- Algorithmic challenges to exploit the hardware capabilities at most.
- Runtime challenges due to the ever growing hardware complexity.

$\Longrightarrow$ MORSE Algorithms: **MORSE-Dense (PLASMA, MAGMA, tile algorithms)**, MORSE-Sparse (PaStiX, MaPHyS), MORSE-Stencil, MORSE-FMM

$\Longrightarrow$ MORSE Runtimes: QUARK, StarPU, **DAGuE**

Matrices Over Runtime Systems at Exascale

$\Longrightarrow$ Mission statement: *"Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems"*.

$\Longrightarrow$ Separation of concerns:

- Algorithmic challenges to exploit the hardware capabilities at most.
- Runtime challenges due to the ever growing hardware complexity.

$\Longrightarrow$ MORSE Algorithms: MORSE-Dense (PLASMA, MAGMA, tile algorithms), **MORSE-Sparse (PaStiX, MaPHyS)**, MORSE-Stencil, MORSE-FMM

$\Longrightarrow$ MORSE Runtimes: QUARK, **StarPU**, DAGuE

Matrices Over Runtime Systems at Exascale

$\Longrightarrow$ Mission statement: *"Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems"*.

$\Longrightarrow$ Separation of concerns:

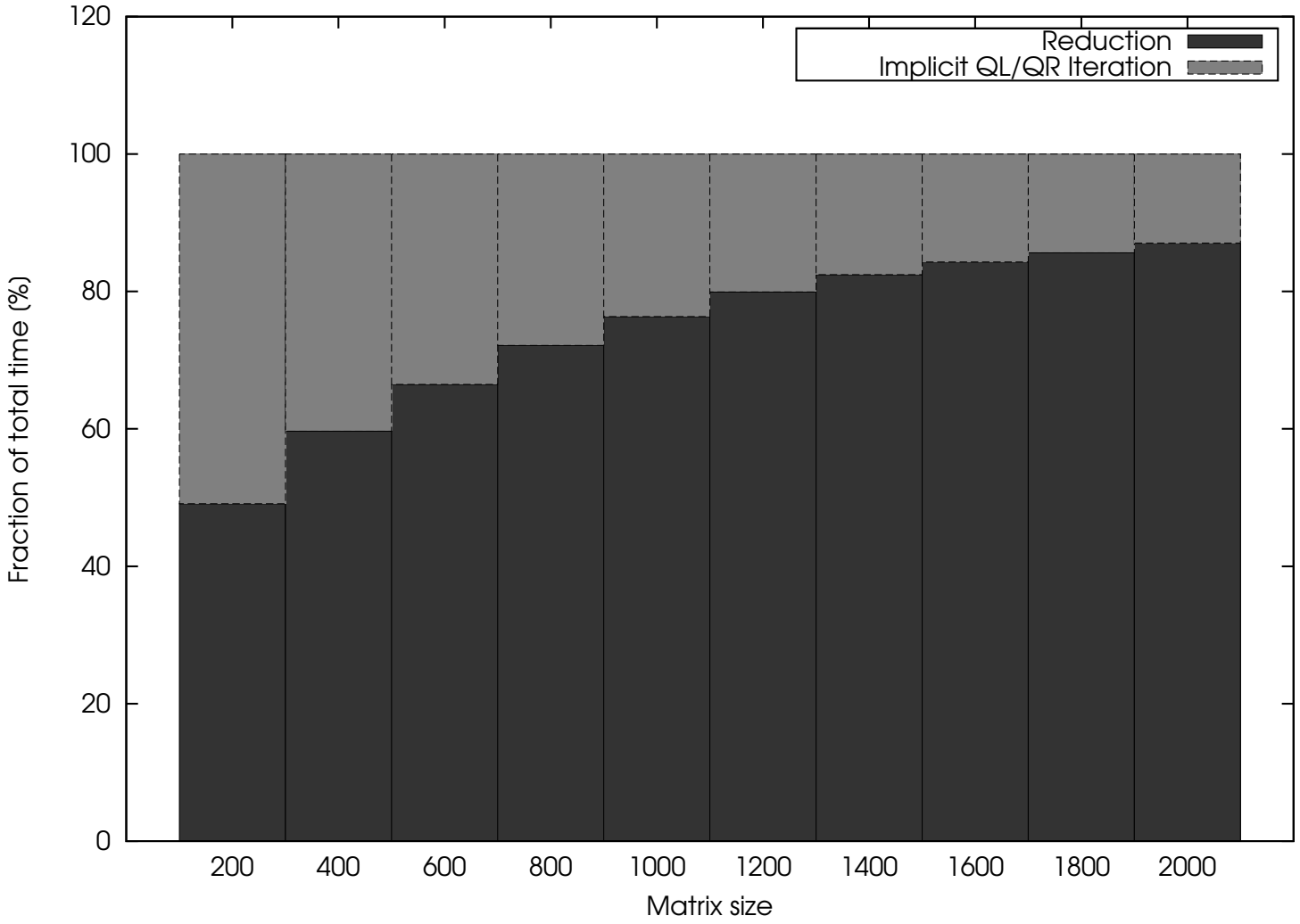
- Algorithmic challenges to exploit the hardware capabilities at most.
- Runtime challenges due to the ever growing hardware complexity.

$\Longrightarrow$ MORSE Algorithms: **MORSE-Dense (PLASMA, MAGMA, tile algorithms)**, MORSE-Sparse (PaStiX, MaPHyS), MORSE-Stencil, MORSE-FMM

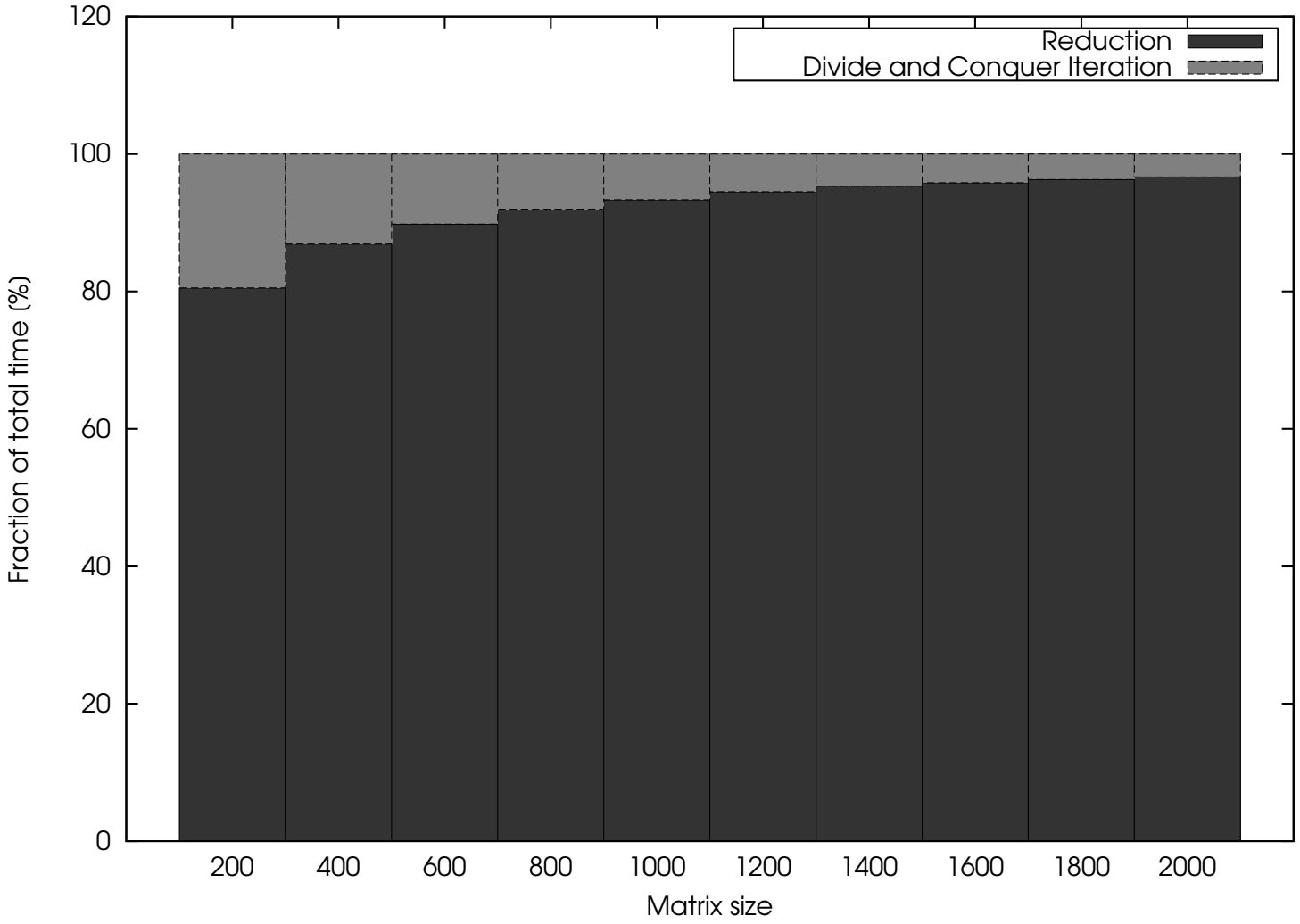
$\Longrightarrow$ MORSE Runtimes: **QUARK**, StarPU, DAGuE

Outline

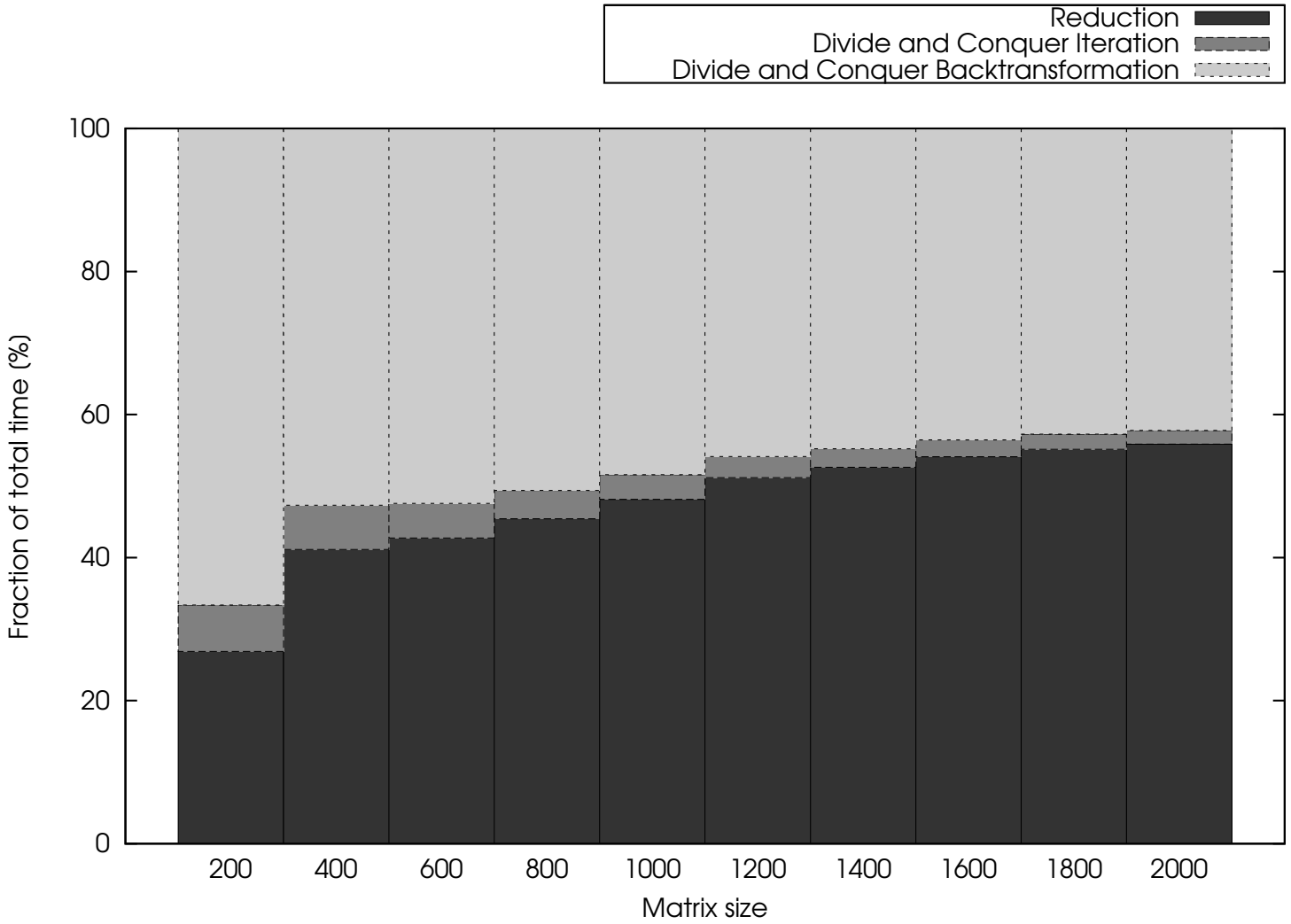
Time Breakdown for TRD and Just Eig-Values (QR)



Time Breakdown for TRD and Just Eig-Vals (D&C)



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

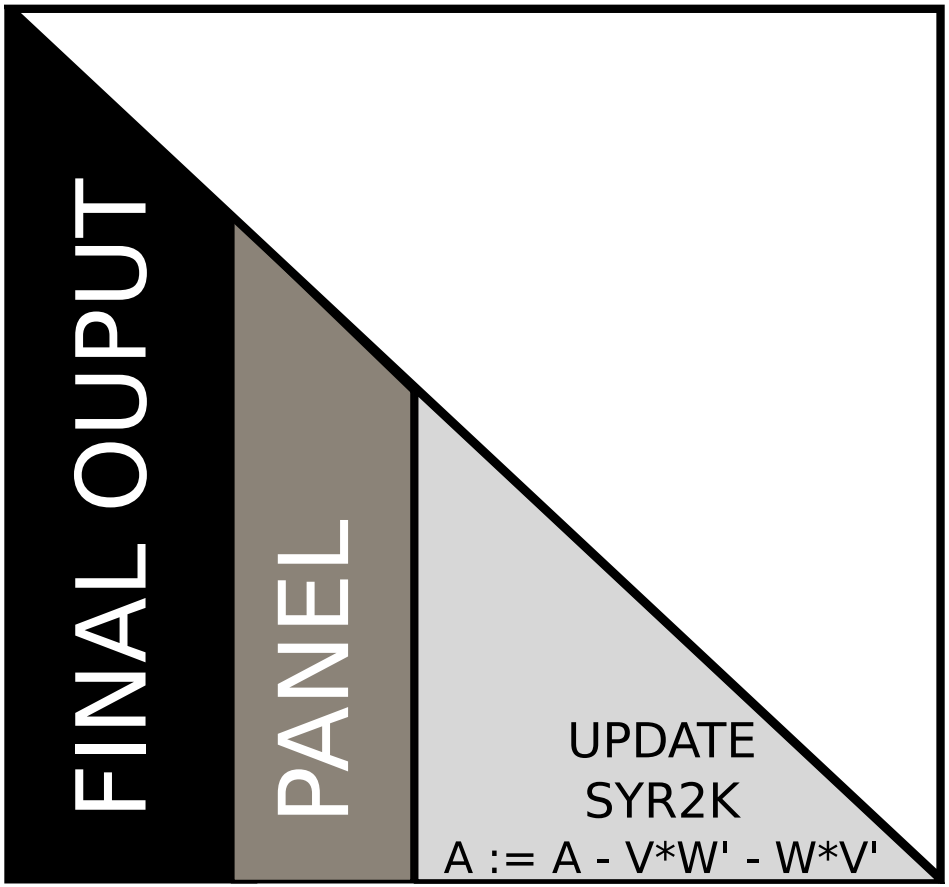


Outline

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

Block Algorithms



Block Algorithms

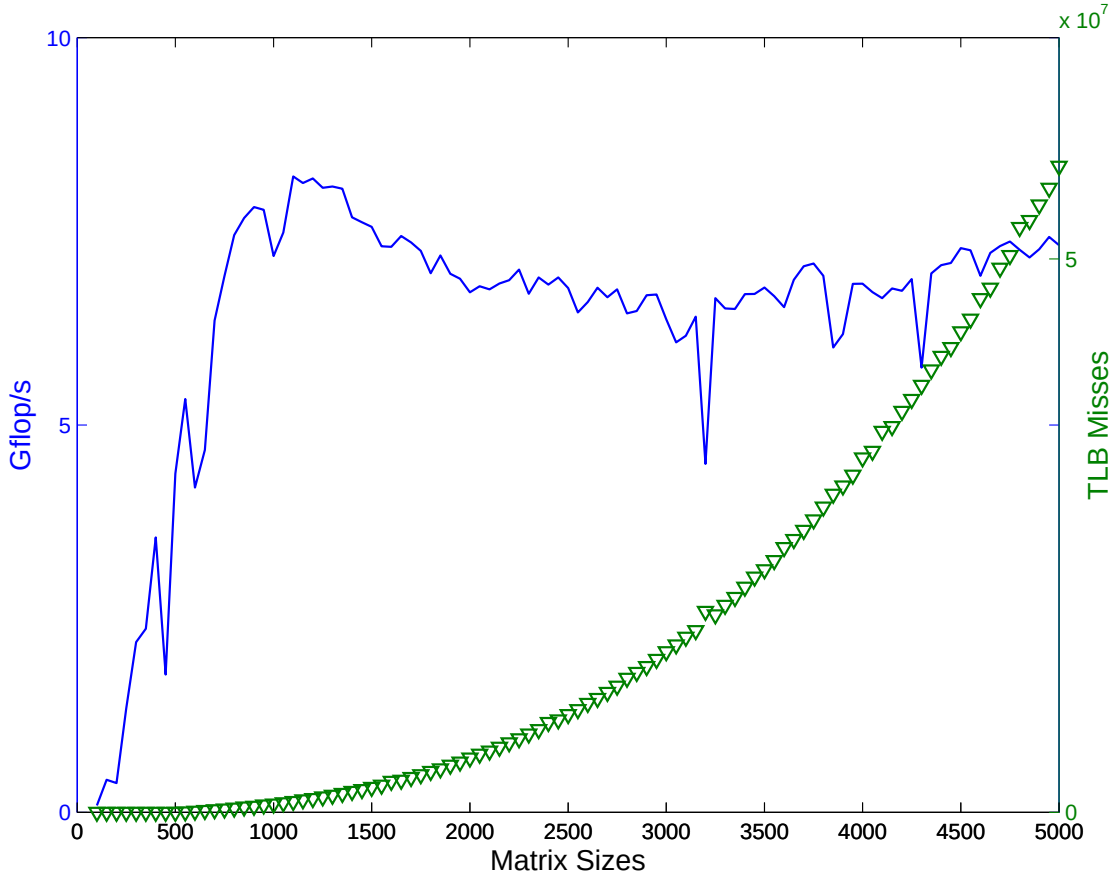


Figure: Performance evaluation and TLB miss analysis of the one-stage LAPACK TRD algorithm with optimized Intel MKL BLAS, on a dual-socket quad-core Intel Xeon (8 cores total).

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
- 2-sided reductions (TRD, BRD, HRD) more challenging than 1-sided factorizations (QR, LU, Cholesky)

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

Tile Algorithms

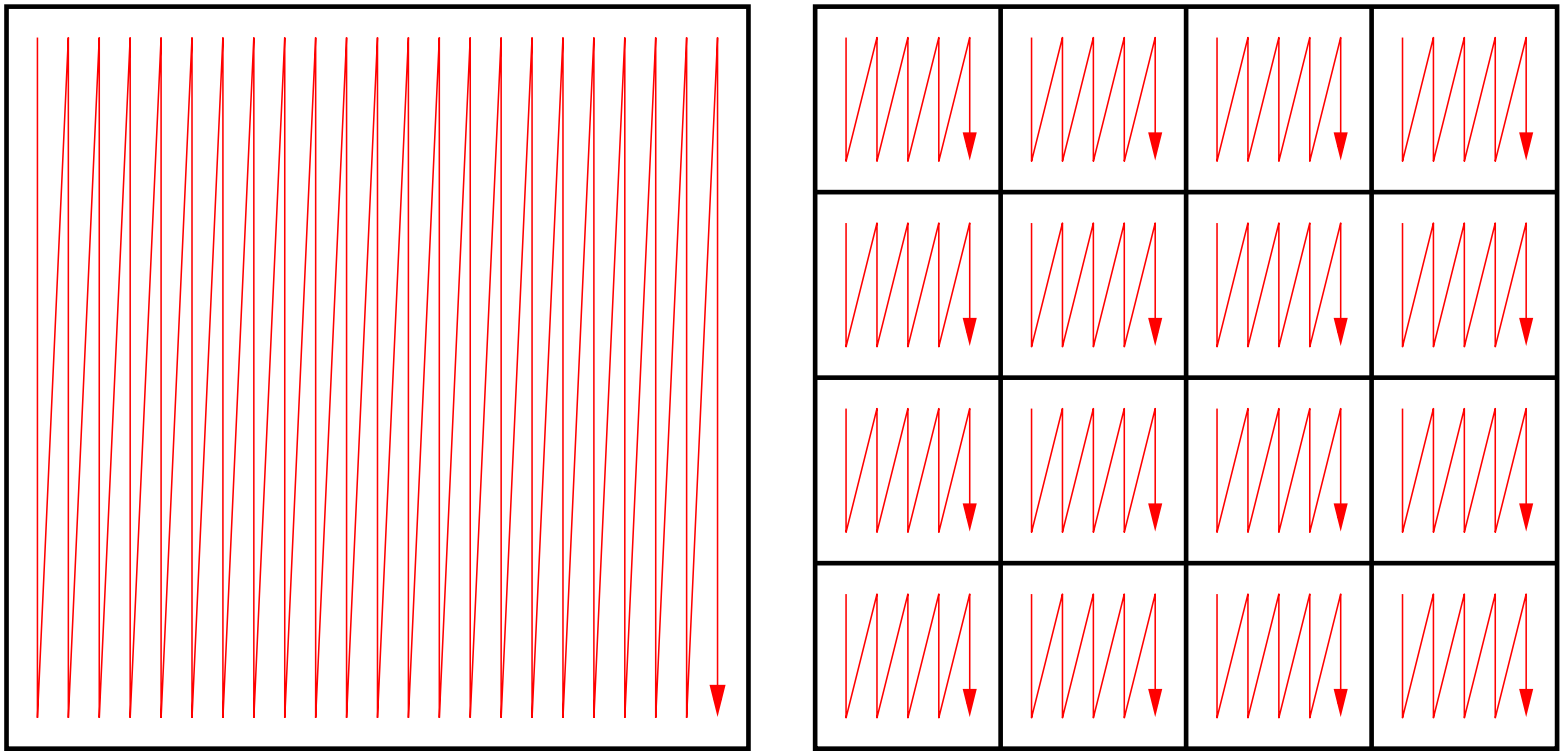


Figure: Translation from LAPACK Layout (column-major) to Tile Data Layout

Outline

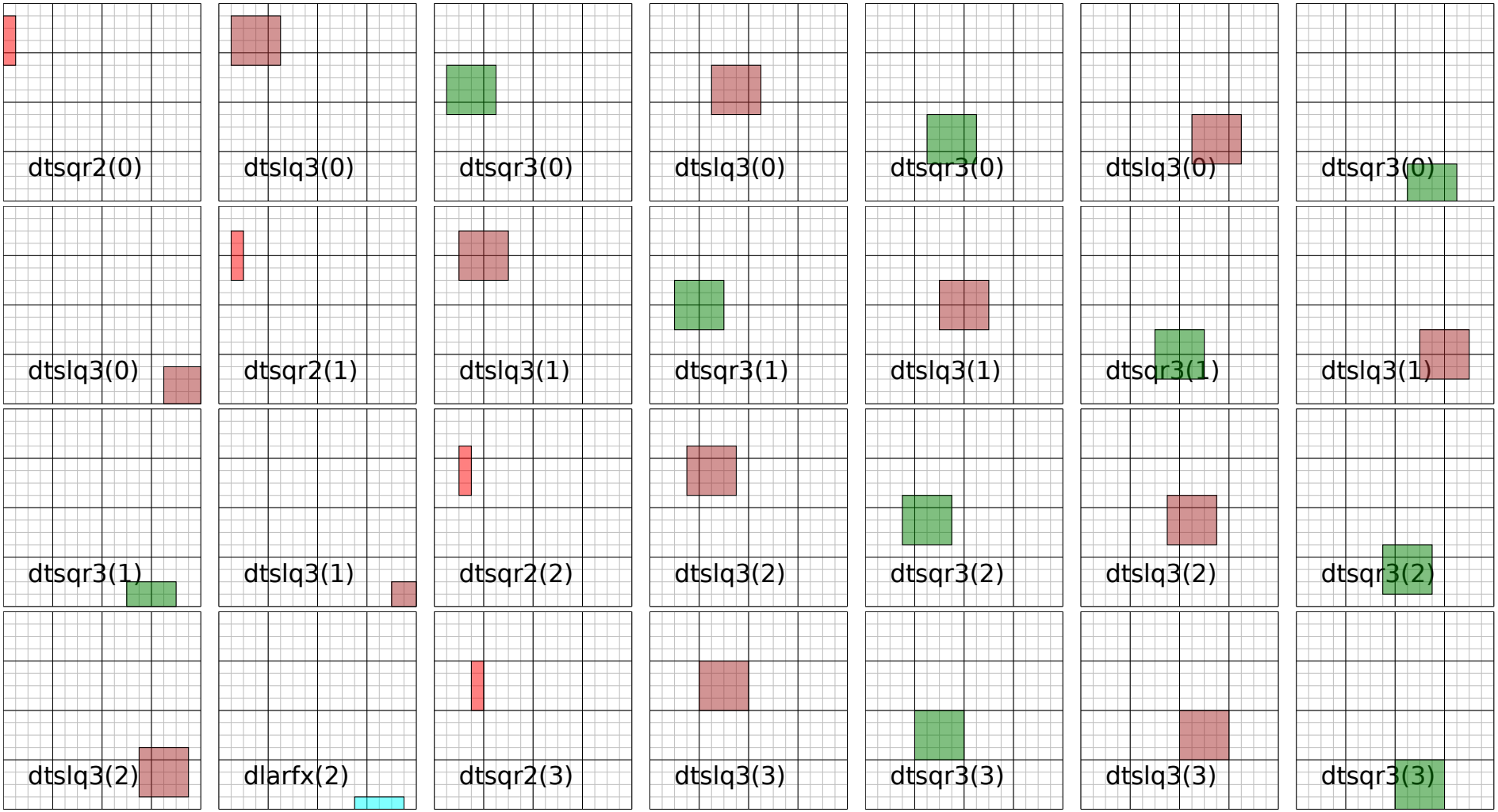
Stage I: Band Reduction

- 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

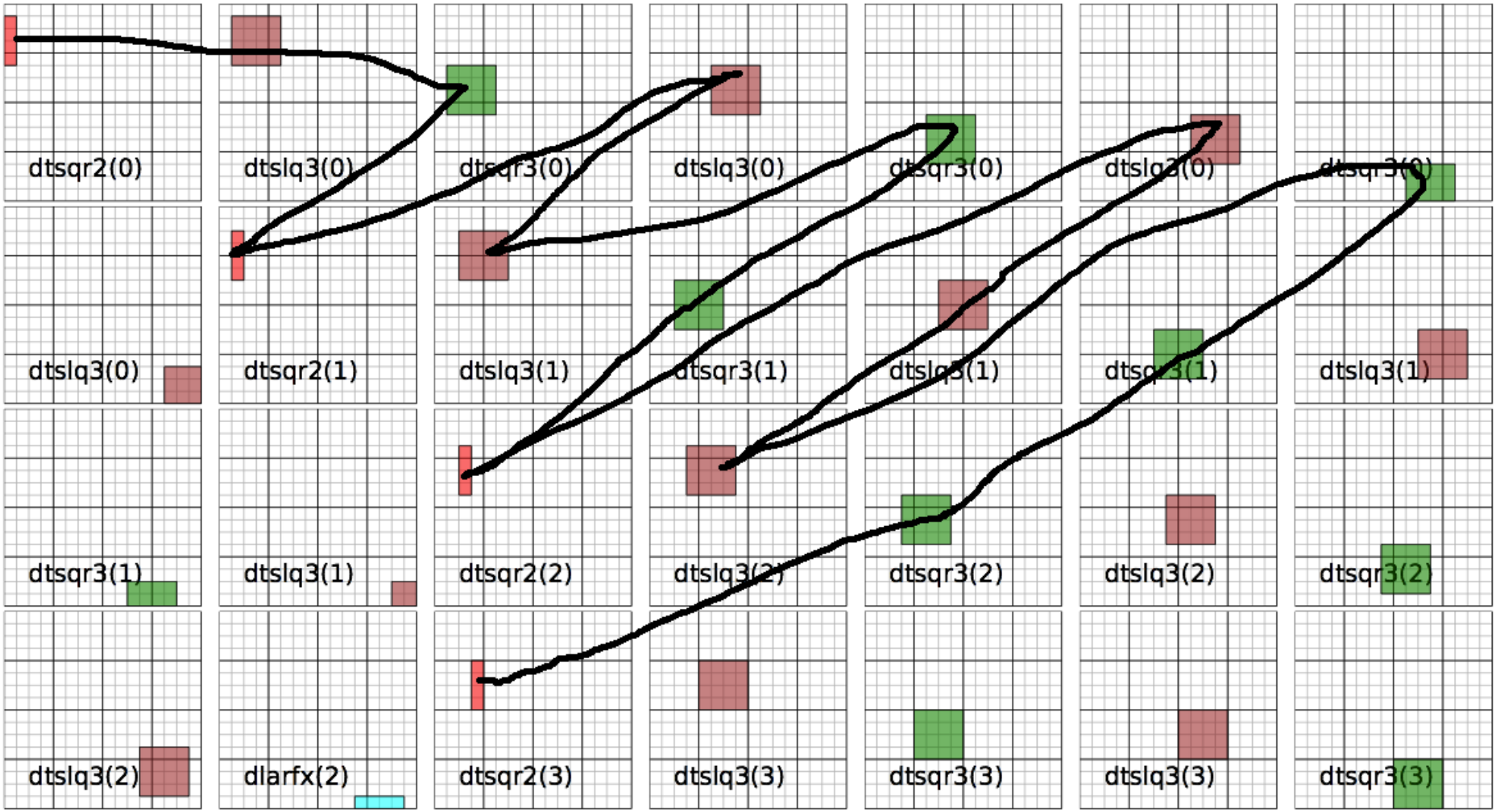
Stage II: Bulge Chasing

- 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.

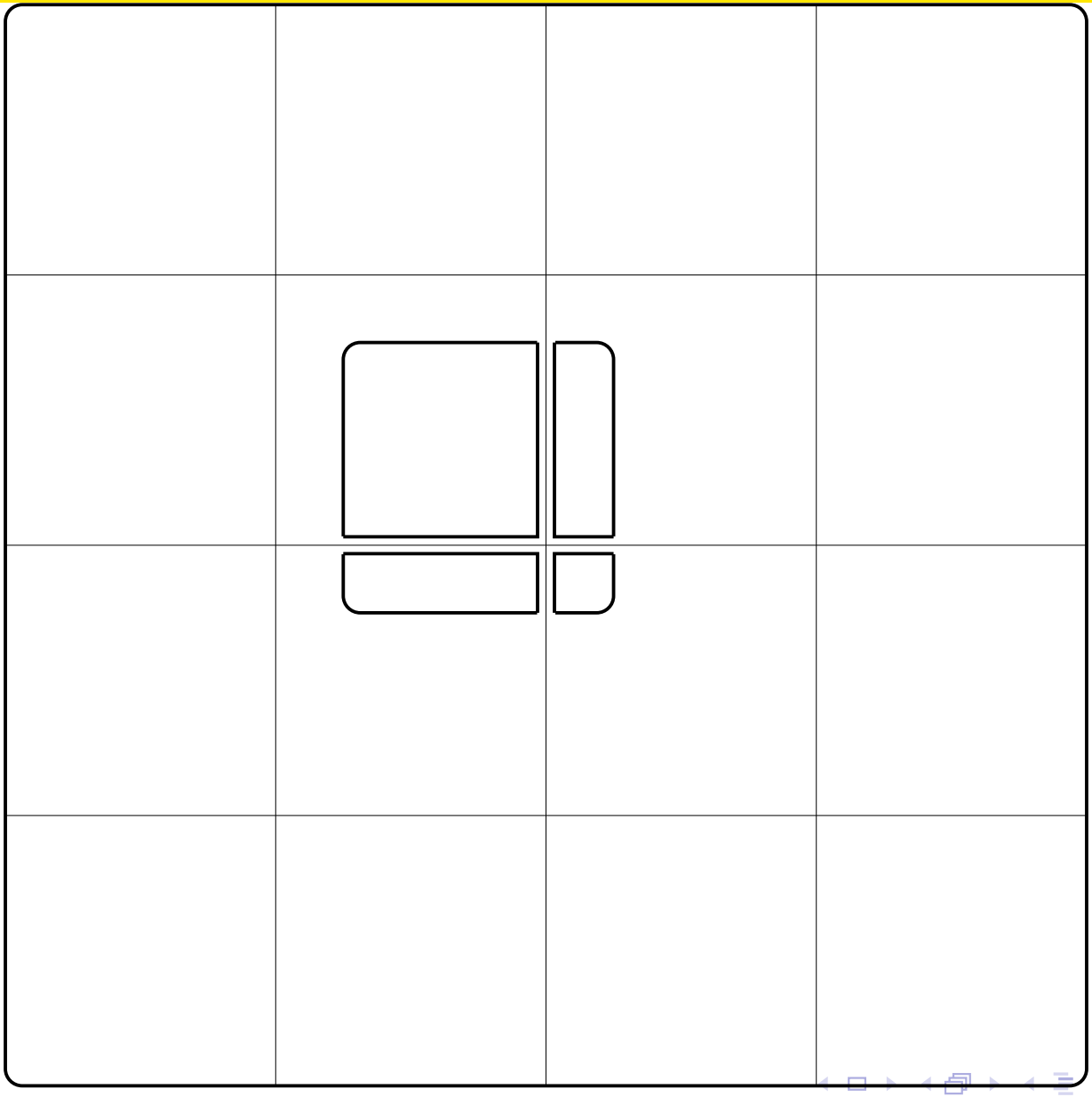
Stage II: Bulge Chasing



Stage II: Bulge Chasing ZigZag



Runtime Translation from Column-major to Tile: DTL



Stage II: Bulge Chasing

- Column-major algorithm running on top of column-major data layout
- Data layout mismatch between both stages
- Need an abstraction layer to reconcile both stage layouts.

Outline

Data Translation Layer

Pro:

- Allows writing algorithm using column-major layout
  - Especially important for bulge chasing which has "shift by column" data access
- Tracks and relays dependences automatically to the runtime scheduler.

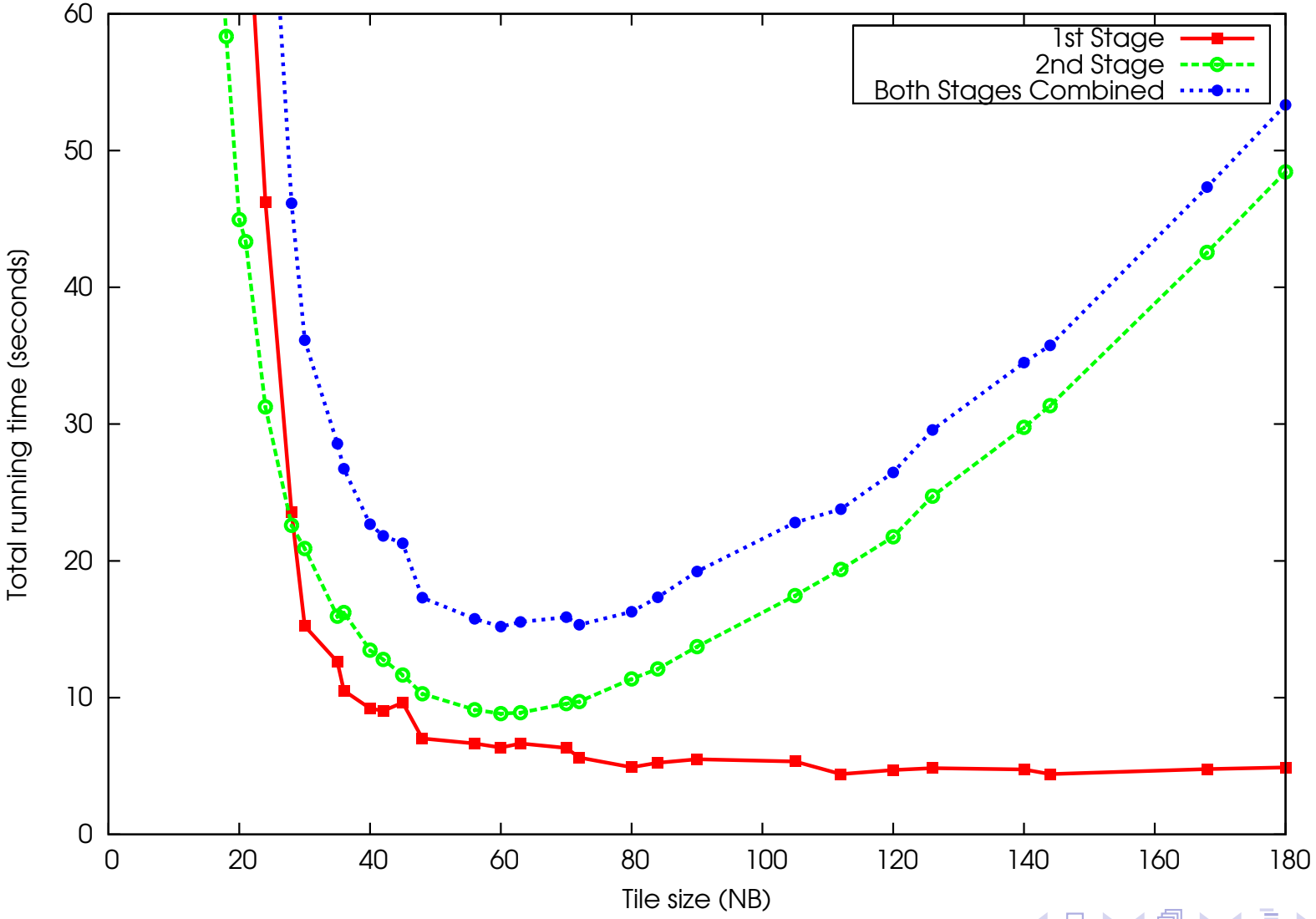
Con:

- Still requires the user to minimize data access area for each kernel call.
  - Minimizing dependences minimizes scheduling overhead and enables more parallelism.

Outline

More Detailed Look at Tile Size (NB) Influence

N=5040



Outline

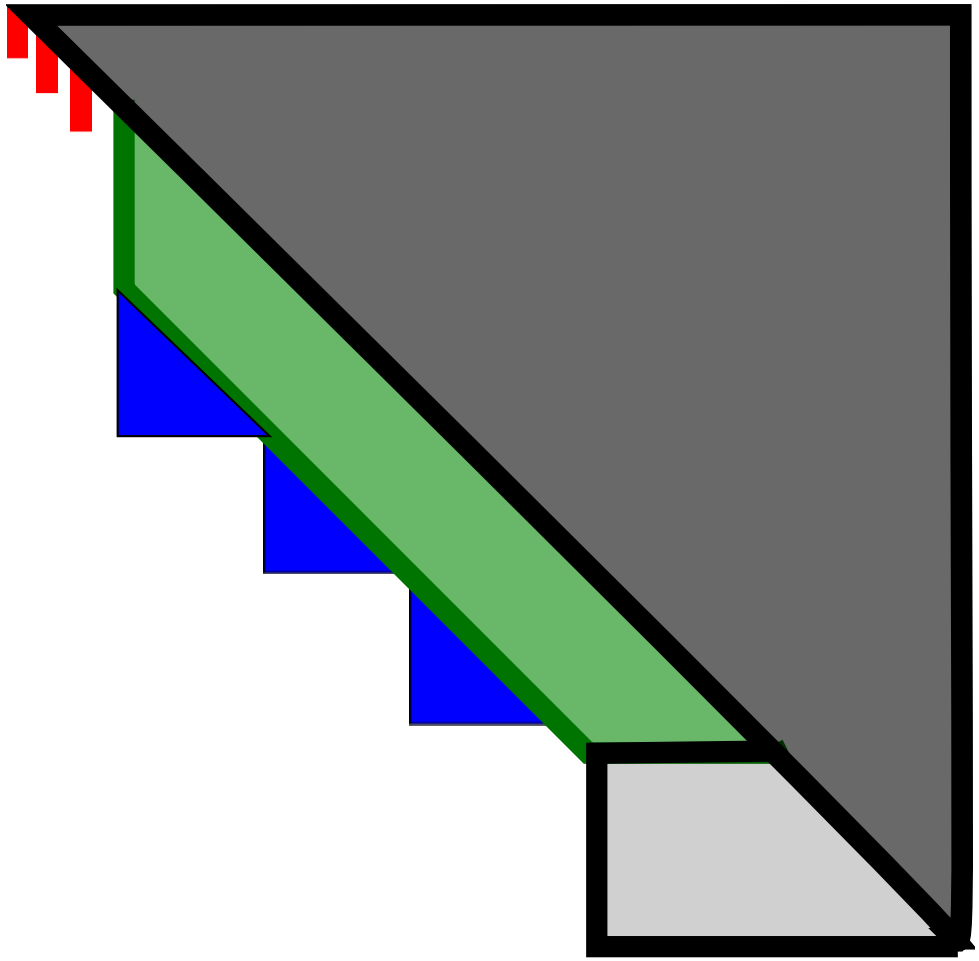
Dynamic Scheduling: QUARK

- Conceptually similar to out-of-order processor scheduling

because it has:

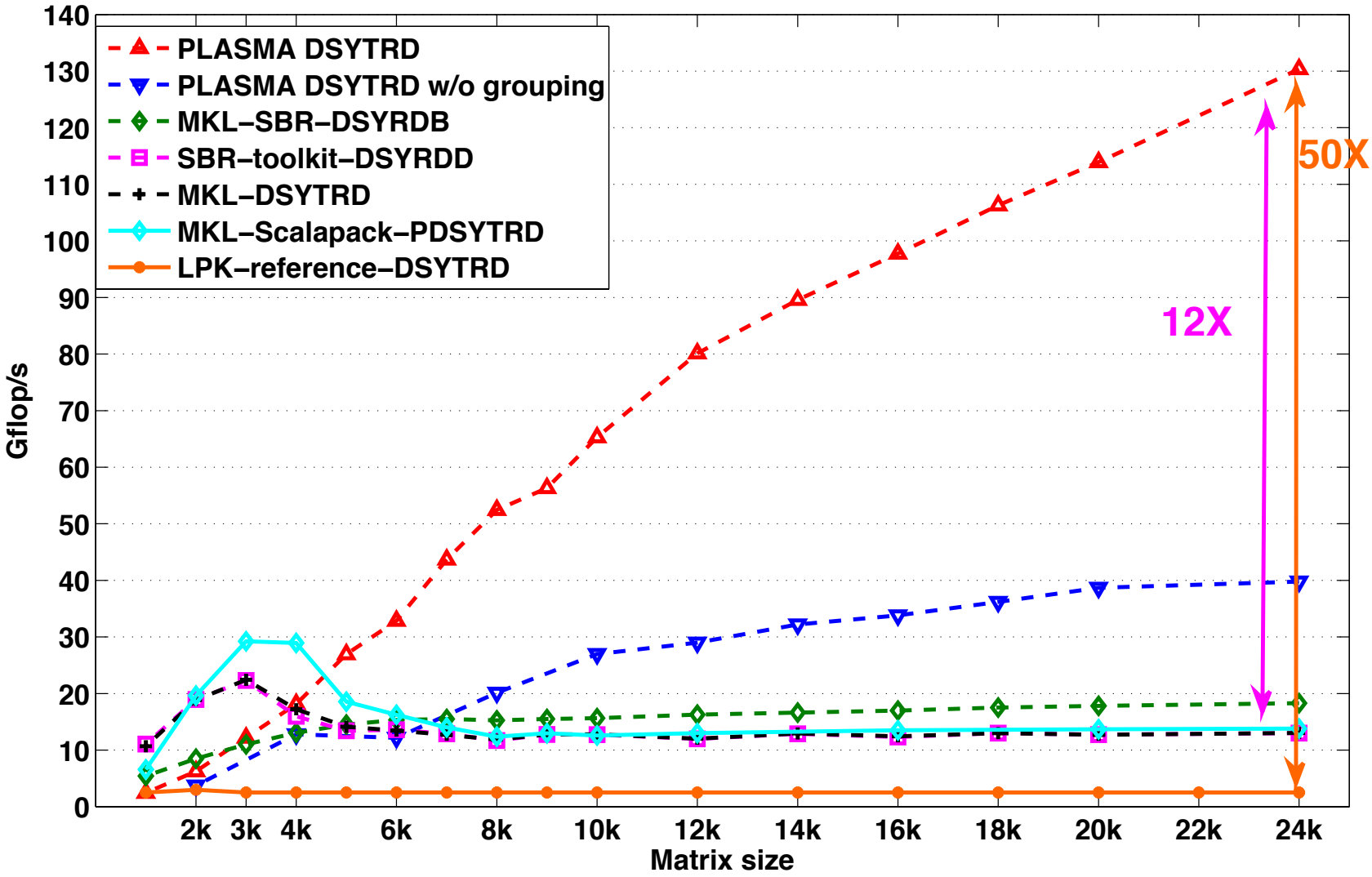
- Dynamic runtime DAG scheduler
- Out-of-order execution flow of tasks
- Task scheduling as soon as dependencies are satisfied
- Overlapping of operations from both stages

Dynamic Scheduling: QUARK



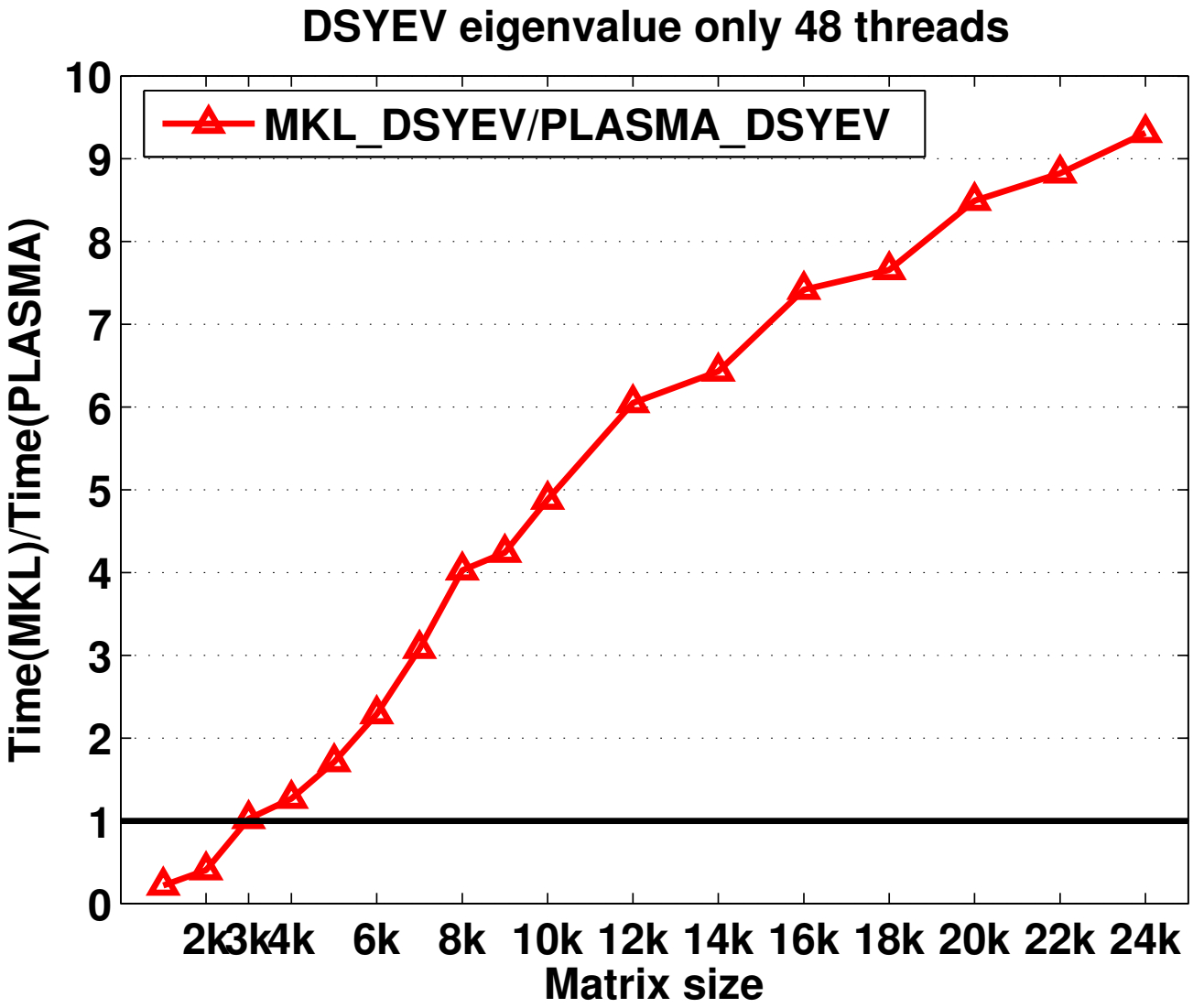
Outline

TRD Performance Results

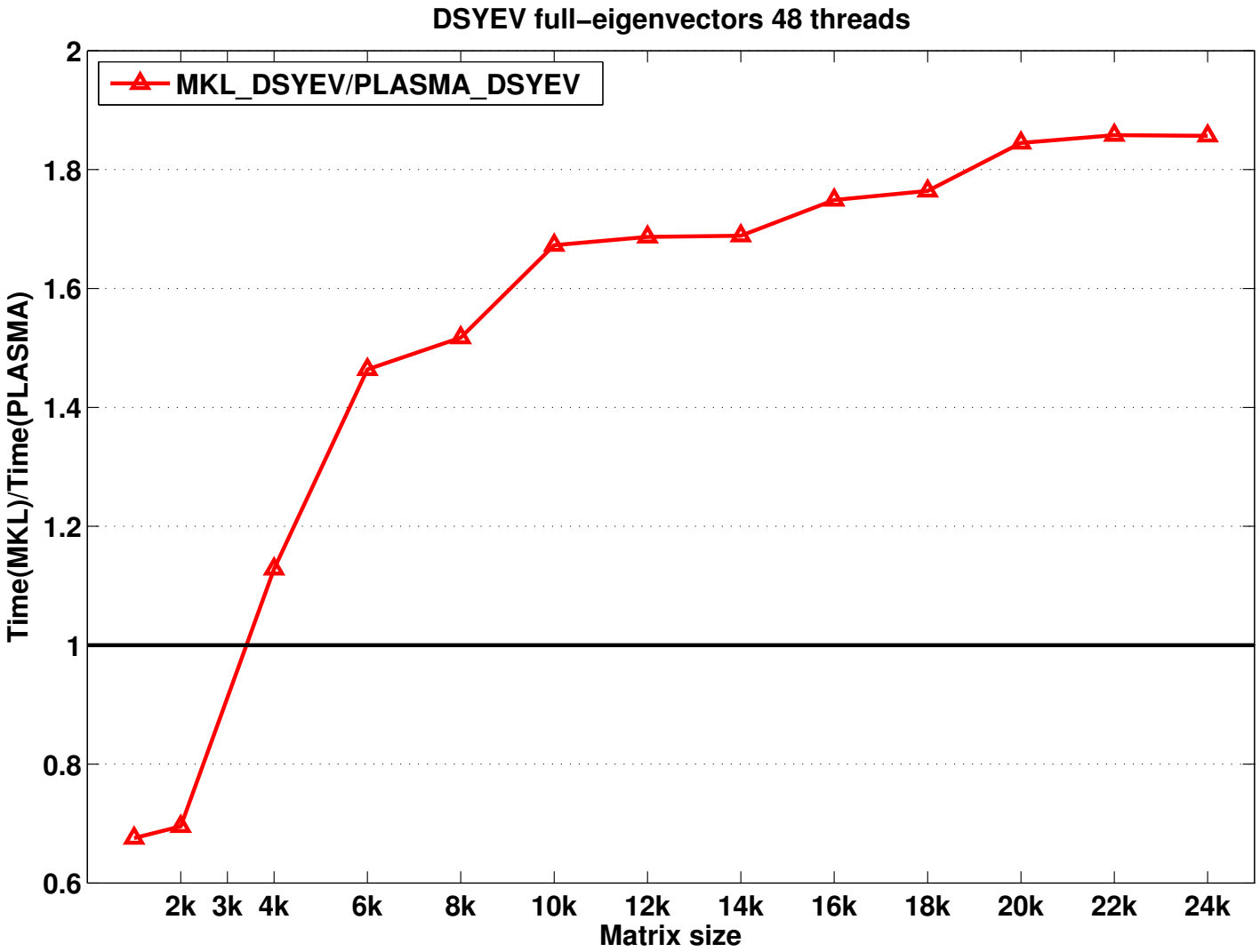


A. Haidar, H. Ltaief and J. Dongarra (SC'11)

Speedup: TRD + Eigenvalue



Speedup: TRD + Eigenvalue + Eigenvector



Outline

Future Work

- Evaluating the performance in the case where only an eigenvalue subset is needed.
- Extension to the Hessenberg reduction (matrix sign function)
- Algorithm favorable for running on heterogeneous hardwares (e.g., StarPU w/ multiple GPUs)
- Implementation on Distributed Environment (DAGuE)

Thank you!

شكرا !