

Efficient and Reliable Algorithms for Challenging Matrix Computations targeting Multicore Architectures and Massive Parallelism

Bo Kågström et al

¹Department of Computing Science and HPC2N
Umeå University, Sweden

PPAM 2011, Torun, Poland, Sept. 11–14, 2011



Matrix computations are ubiquitous!



ABB robot



Saab JAS Gripen



Ibanez



Saab 9-3: a real car!?!

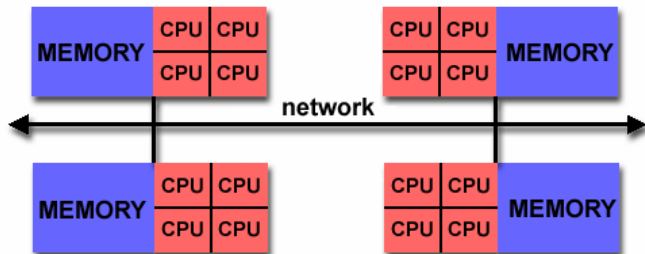
Large-scale matrix computations

- **Fundamental** and **ubiquitous** in computational science and its vast application areas
- **Library software**: optimized building blocks of fundamental operations
 - **BLAS**, (Sca)**LAPACK**, SLICOT (see also NETLIB)
 - NAG, ESSL and other vendors
 - **Portability, robustness and efficiency**
- Continuing **demand for new and improved algorithms and software** along with the computer evolution

Hybrid distributed memory HPC systems

- Target parallel computers have **SMP-like/multicore nodes** and a **fast interconnect**

Programming model is a combination of **message passing** and **multithreading**



- Architecture evolution:** more cores per node, larger node memories, several levels of caches, hardware accelerators (GPGPU, FPGA, Cell)

Minimize data transport in memory hierarchies

- **Some challenges:**
 - Memory bandwidth (off-chip) bottleneck
 - Scaling to many cores
 - Heterogeneous multicores
 - Granularity of tasks/threads
 - Synchronization avoidance
 - Communication avoidance
- **Key to performance:** understand the algorithm and architecture interaction
- **Maximize computational intensity and data reuse!**
(CI = #flops/#mem refs, level 3 is n^3/n^2)
- **Hierarchical blocking:**
 - Explicit (multilevel) blocking
 - Recursive blocking
 - *Spatial* and *temporal* locality



Objective: To build a **European vision and roadmap** to address the **challenges** of the new generation of massively parallel systems composed of **millions of heterogeneous cores** which will provide multi-Petaflop performances in the next few years and Exaflop performances in 2020. (Co-funded by the European Commission.)

Exa = 10^{18}

IESP (**International Exascale Software Project**) co-funded by DOE and NSF in USA.

WP3: Application Grand Challenges

WP Chair: Stéphane Requena (GENCI)

WG 3.1 Industrial and Engineering Applications

WG 3.2 Weather, Climatology and Earth Sciences

WG 3.3 Fundamental Sciences (Chemistry, Physics)

WG 3.4 Life Science and Health

Chair

Philippe Ricoux (TOTAL)

Giovanni Aloisio (ENES-CMCC)

Godehard Sutmann (CECAM)

Modesto Orozco (BSC)

WP4: Enabling Technologies for Exaflop Computing

WP Chair: Bernd Mohr (Jülich)

WG 4.1 Hardware Roadmaps, Links with Vendors

WG 4.2 Software Eco-system

[WG 4.3 Numerical Libraries, Software and Algorithms](#)

WG 4.4 Scientific Software Engineering

Chair

Herbert Huber (STRATOS-LRZ)

Franck Cappello (INRIA-UIUC)

[Iain Duff \(STFC-RAL and CERFACS\)](#)

Mike Ashworth (STFC-DL)

EESI Working Group 4.3 – Experts

Iain Duff	STFC/CERFACS	UK	Sparse Linear Algebra
Andreas Grothey	University of Edinburgh	UK	Cont & Stoch Optimization
Patrick Amestoy	ENSEEIH-IRIT, Toulouse	FR	Sparse Direct Methods, Solvers
Peter Arbenz	ETH Zürich	CH	Eigenvalues, HPC
Jack Dongarra	Tennessee/Manchester	UK/US	HPC, Numerical LA
Salvatore Filippone	Università di Roma	IT	Numerical Software
Mike Giles	University of Oxford	UK	GPU, CFD/Finance
Luc Giraud	INRIA Bordeaux	FR	Iterative & Hybrid Methods
Thorsten Koch	Zuse-Institut Berlin	DE	Combinatorial Optimization
Bo Kågström	Umeå University	SE	HPC, Dense Linear Algebra
Karl Meerbergen	K.U. Leuven	BE	Preconditioners, ExaScience Lab
Volker Mehrmann	TU Berlin	DE	Linear Algebra, HP Applications
Gerard Meurant	ex-CEA	FR	HPC, PDE solution
François Pellegrini	LaBRI Bordeaux	FR	Partitioning
Julius Žilinskas	Vilnius University	LT	Global Opt, Meta-heuristics

Main areas covered:

- Dense linear algebra
- Graph and hypergraph partitioning
- Sparse direct methods
- Iterative methods for sparse matrices
- Eigenvalue problems, model reduction
- Optimization
- Control of complex systems
- Structured and unstructured grids

Much interdependence between areas.

Importance of [also working at Tera- and Petascale levels](#).

Dense linear algebra

- **Fundamental building blocks** for much of numerical computations.
- **BLAS** still basis but needs rethinking to advance on current multi-threaded BLAS. **GEMM** still the archetypal building block.
- Reduce data movement and improve locality of reference by using **block data structures** and conversions between different structures.
- *One-sided* (matrix factorizations) and *two-sided* (eigenreduction, solving matrix equations).
- **Multidimensional array** (tensor) decompositions.
- **Iterative methods for dense** problems: linear systems, eigenvalue problems (e.g., Krylov-based, see also Iterative methods ..)

Dense linear algebra (2)

- Moving from **fork-join** to use of **task dags** to reduce synchronization overheads.
- Use of **mixed-precision arithmetic** important.
- **Communication avoidance** and hiding.
- **Reproducibility** of results. A posteriori error estimation.
- Very good European presence in this area. Involved in BLAS, (Sca)LAPACK, SLICOT, ...

Cross-cutting issues:

- Fault tolerance
- Autotuning and performance optimization
- Energy aware algorithms

- Motivation and background – **challenges!**
- **Topic I:** Parallel and cache-efficient in-place matrix storage format conversion
- One-sided vs. **two-sided** matrix transformations
- **Topic II:** Parallel two-stage reduction to Hessenberg form on shared memory architectures
- **Topic III:** Parallel QR and QZ multishift algorithms with advanced deflation strategies
- Computational experiments

Topic I – matrix storage format conversions

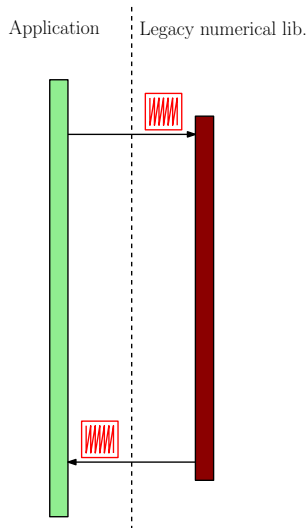
- Why matrix data layout conversions? Why in-place?
- Blocked storage formats
- In-place-conversion via in-place matrix transpositions
- Conversion graph of matrix storage formats
- Parallelization and cache-efficiency issues
- Overhead and performance results

Fred Gustavson, Lars Karlsson, and Bo Kågström, Parallel and Cache-Efficient In-Place Matrix Storage Format Conversion, *ACM Trans. Math. Software* 2011 (accepted).

Application using legacy HPC library

Matrix storage format: CM

- Column-major (**CM**) and row-major (**RM**) storage formats are typically used by compilers
- BLAS, LAPACK, ScaLAPACK, etc. assume that inputs are in **CM** format
- **Blocks are scattered in memory**
- Remedy: **Use blocked data layouts internally!**



Blocked storage formats

Standard formats

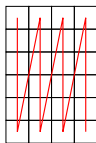
- **CM** Column-Major
- **RM** Row-Major
- **Inefficient block access**

Blocked formats

- **CCRB** Column-Column RB
- **CRRB** Column-Row RB
- **RCRB** Row-Column RB
- **RRRB** Row-Row RB
- **Blocks** are **stored contiguously** in memory

(RB = Rectangular Block)

CM



RM



CCRB



CRRB



RCRB

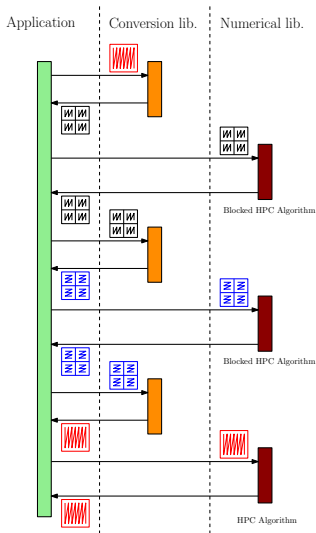


RRRB

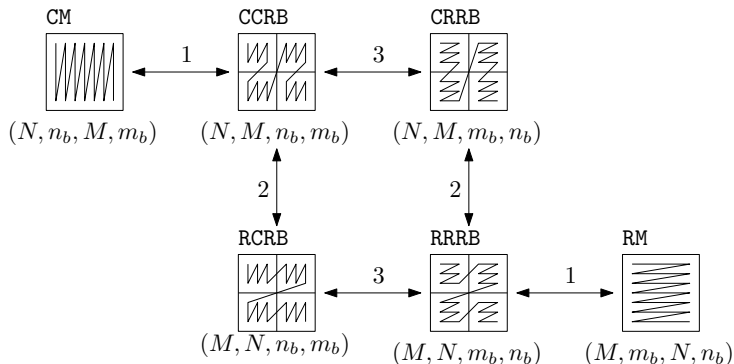


Application with mixed use of HPC library routines

Matrix storage format conversions: CM \rightarrow CCRB \rightarrow RRRB \rightarrow CM



Stage graph for in-place conversion



Each stage consists of **one or several independent blocked in-place transpositions** (e.g., Stage 2: one with chunk size $L = m_b \cdot n_b$)

Conversion using in-place transposition

Permutations and cycles

- Move the element (i, j) from location $k = i + jm$ to $\hat{k} = j + in$
- In-place transposition is a *permutation*
- Every permutation can be decomposed into a set of *disjoint cycles*

Example (cycles of the 5×3 transposition permutation)

$(0)(1\ 3\ 9\ 13\ 11\ 5)(7)(2\ 6\ 4\ 12\ 8\ 10)(14).$

In other words: locations 0, 7, and 14 are fixed, 1 goes to 3 which goes to 9, ..., 5 which goes back to 1, and so on

- $a = \text{reshape}(A, 15, 1)$ indexed from $0:m*n-1$
- $a([3\ 9\ 13\ 11\ 5\ 1]) = a([1\ 3\ 9\ 13\ 11\ 5])$
- $a([6\ 4\ 12\ 8\ 10\ 2]) = a([2\ 6\ 4\ 12\ 8\ 10])$
- $A = \text{reshape}(a, 3, 5)$

Conversion using in-place transposition

Permutations and cycles

- Move the element (i, j) from location $k = i + jm$ to $\hat{k} = j + in$
- In-place transposition is a *permutation*
- Every permutation can be decomposed into a set of *disjoint cycles*

Example (cycles of the 5×3 transposition permutation)

$(0)(1\ 3\ 9\ 13\ 11\ 5)(7)(2\ 6\ 4\ 12\ 8\ 10)(14)$.

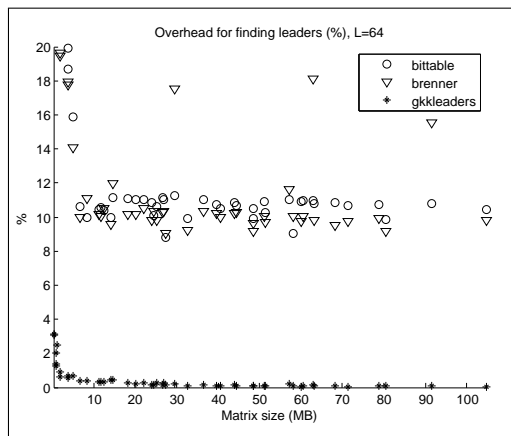
In other words: locations 0, 7, and 14 are fixed, 1 goes to 3 which goes to 9, ..., 5 which goes back to 1, and so on

- Cycle shifting is straightforward
- **Main problem:** find the cycle structure efficiently (cycle leaders, cycle lengths)
- Cache efficiency via blocked in-place transpositions

Compared three algorithms that find cycle leaders:

- **GKKLeaders**: our new algorithm
 - finds leaders without searching
 - number theoretical properties of the prime factorization of $q = m \cdot n - 1$
- **Brenner**: based on Brenner's ACM Algorithm 467
 - searching combined with number theory to prune the search
- **Bittable**: uses a full bittable (simplest possible approach)
 - traverses every cycle while making marks in a bittable

Cycle-following algorithms: overhead



- Our algorithm **GKKLeaders** has a negligible overhead when shifting **blocks of 64 elements** on large matrices.
- The advantage comes from not having to traverse any cycle to determine its leader and/or length.

Shared memory parallelization – three levels

The **conversion cost** is **dominated by moving the data**.
Finding cycle leaders need not be parallelized (minor cost).

(I) Independent in-place transpositions

- Large granularity
- Load balancing is straightforward

(II) Independent cycles within one transposition

- Cycles (different length) can be shifted independently
- Load balancing required
- Cycle structure obtained cheaply ahead (GKK algorithm)

(III) Splitting long cycles

- Long cycle split into several pieces (some workspace)
- Pieces shifted independently after initial synchronization

Data format conversion: performance

Normalized execution times (*ns/element*) measured on Akka

From \ To	CM	CCRB	CRRB	RCRB	RRRB	RM
CM		2.2	4.8	4.1	6.6	8.9
CCRB	2.3		2.5	1.8	4.4	6.7
CRRB	4.8	2.6		4.4	1.9	4.1
RCRB	4.1	1.8	4.4		2.6	4.8
RRRB	6.7	4.4	1.9	2.6		2.3
RM	8.9	6.6	4.1	4.8	2.3	

The average time per stage ($t_{\text{copy}} = 3.02$ and $t_{\text{scale}} = 1.62$)

From \ To	CM	CCRB	CRRB	RCRB	RRRB	RM
CM		2.2	2.4	2.1	2.2	2.2
CCRB	2.3		2.5	1.8	2.2	2.2
CRRB	2.4	2.6		2.2	1.9	2.1
RCRB	2.1	1.8	2.2		2.6	2.4
RRRB	2.2	2.2	1.9	2.6		2.3
RM	2.2	2.2	2.1	2.4	2.3	

In-place matrix storage format conversion highlights

- In-place conversion via blocked in-place transpositions
- GKK: new algorithm for a priori determination of leaders and cycle lengths
- Parallelism for moving data at three levels
- Generalizing to ***arbitrary matrix dimensions*** by isolating submatrices
- ***Software*** is available

Topic II – parallel two-stage Hessenberg reduction

- $Q^T A Q = H$; Why two stages? Trade-off!
- **Stage 1:** $A \rightarrow H_r$ in (block) r -Hessenberg form
 - Dynamic scheduling of tasks and atoms with precedences
- **Stage 2:** $H_r \rightarrow H$ in Hessenberg form
 - Adaptive coarse-grained load balancing; one level of look-ahead “hides” sequential bottleneck
- Computational experiments

Lars Karlsson and Bo Kågström, Efficient Reduction from Block Hessenberg Form to Hessenberg Form using Shared Memory, In *PARA 2010, LNCS* Springer 2011 (to appear).

Lars Karlsson and Bo Kågström, Parallel Two-Stage Reduction to Hessenberg Form using Dynamic Scheduling on Shared-Memory Architectures, *Parallel Computing*, Elsevier 2011 (avail. online).

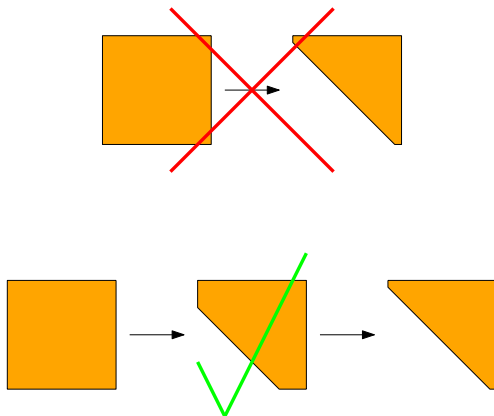
Reduction to **Hessenberg form**:

$$Q^T A Q = H = \begin{array}{|c} \hline & & & & \\ & \diagdown & & & \\ & & \diagdown & & \\ & & & \diagdown & \\ & & & & \diagdown \\ \hline \end{array}$$

- Blocking via the compact WY representation
- 80% of the floating point operations are fast Level 3 BLAS
- **20% of the floating point operations are slow Level 2 BLAS**
- Poor scalability on multicore processors

Hessenberg reduction – two stages instead of one

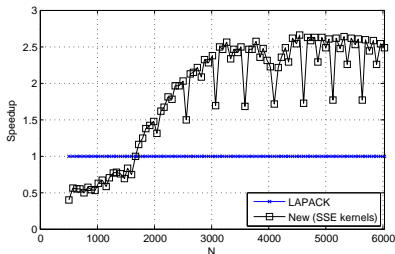
We present fast, parallel, and cache-efficient algorithms for two-stage reduction to Hessenberg form on shared-memory architectures.



Speedup

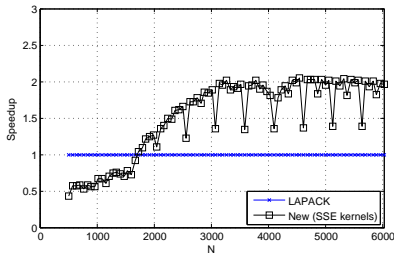
2x Intel Xeon L5420 (8 cores)

Only H



Up to 2.5x speedup over
LAPACK

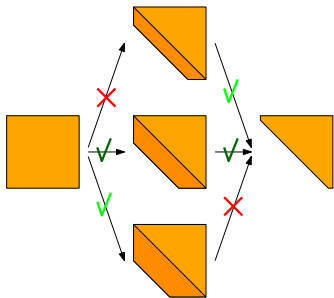
Both H and Q



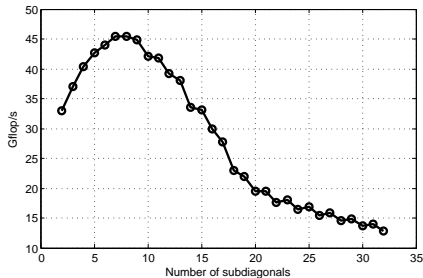
Up to 2.0x speedup over
LAPACK

Trade off between the two stages

High performance obtained when the block Hessenberg form has **relatively few subdiagonals** (but not too few).

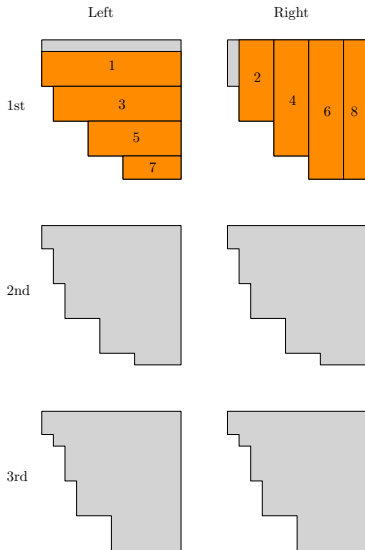


Stage 2



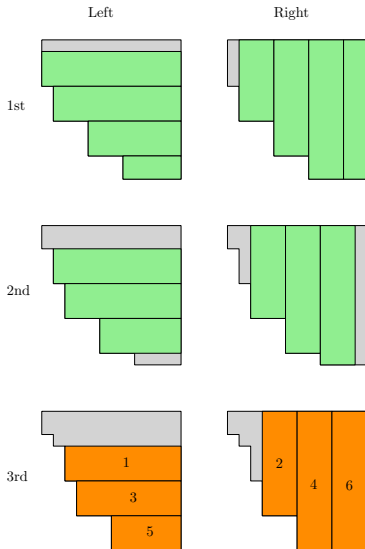
Stage 2: Block Hessenberg to Hessenberg reduction

Access pattern of Basic Algorithm



Stage 2: Block Hessenberg to Hessenberg reduction

Access pattern of Basic Algorithm



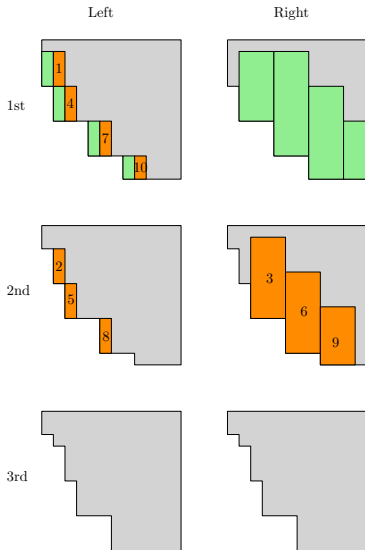
Stage 2: The new cache-efficient algorithm

Access pattern



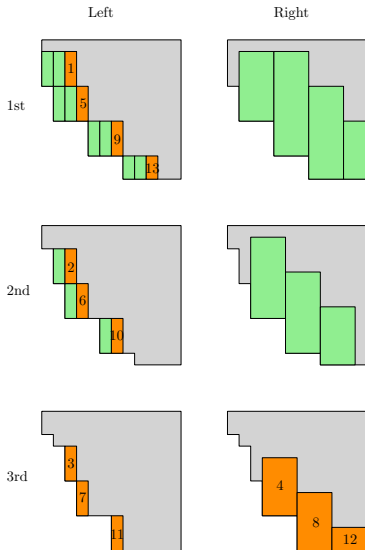
Stage 2: The new cache-efficient algorithm

Access pattern



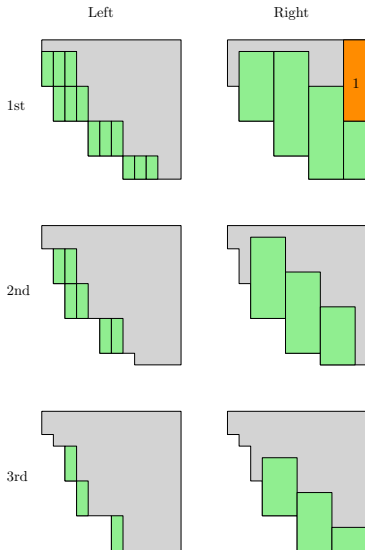
Stage 2: The new cache-efficient algorithm

Access pattern



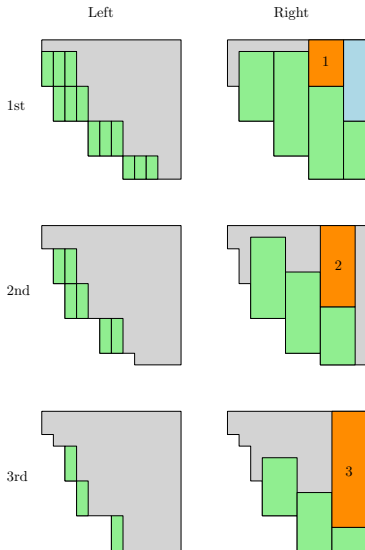
Stage 2: The new cache-efficient algorithm

Access pattern



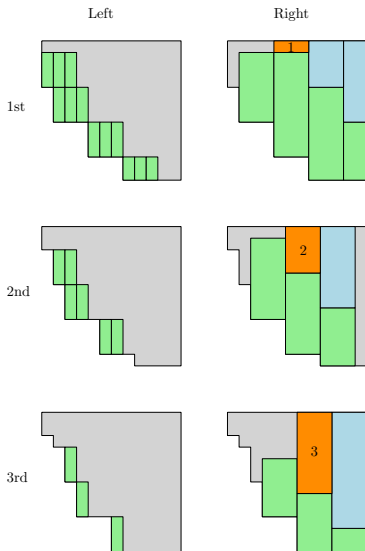
Stage 2: The new cache-efficient algorithm

Access pattern



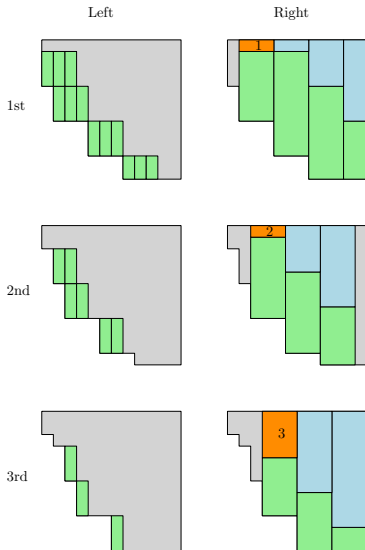
Stage 2: The new cache-efficient algorithm

Access pattern



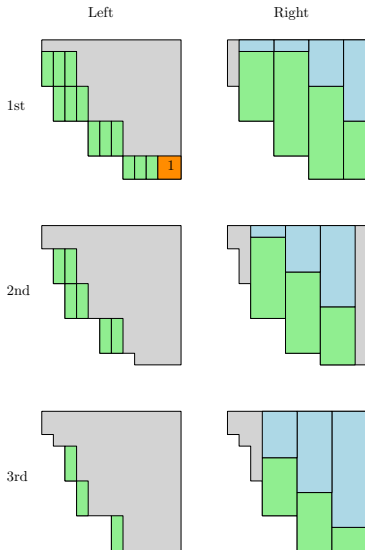
Stage 2: The new cache-efficient algorithm

Access pattern



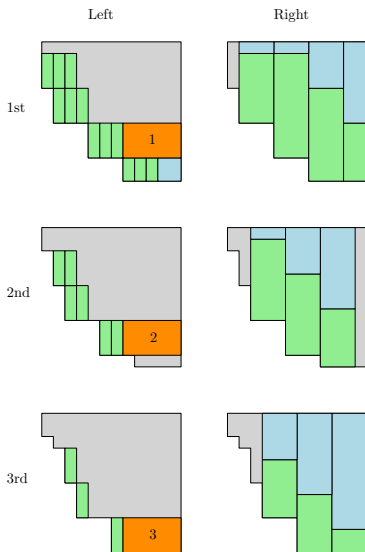
Stage 2: The new cache-efficient algorithm

Access pattern



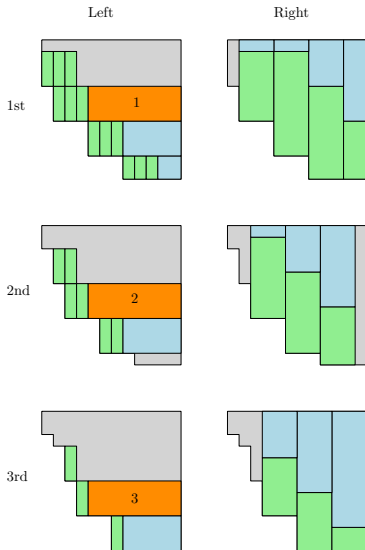
Stage 2: The new cache-efficient algorithm

Access pattern



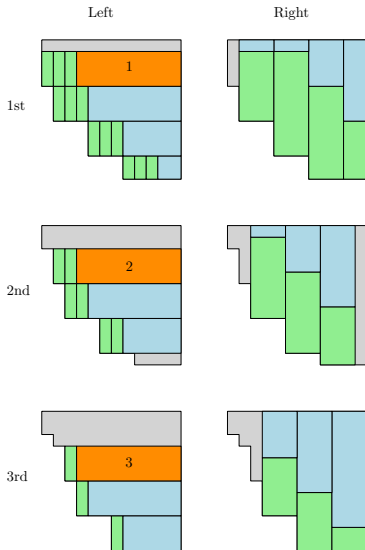
Stage 2: The new cache-efficient algorithm

Access pattern



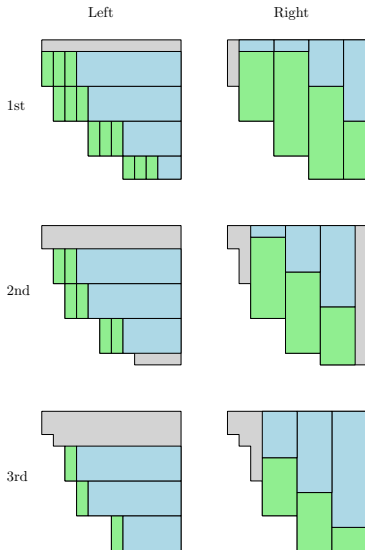
Stage 2: The new cache-efficient algorithm

Access pattern



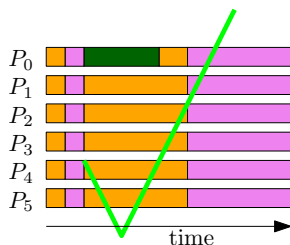
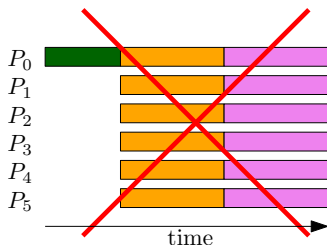
Stage 2: The new cache-efficient algorithm

Access pattern



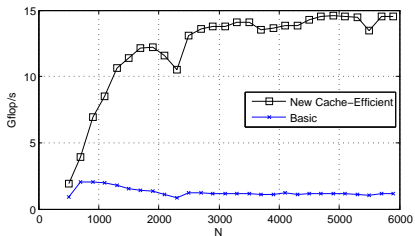
Stage 2: Look-ahead is possible

The **look-ahead** technique can be applied in the second stage despite its **two-sided** nature.



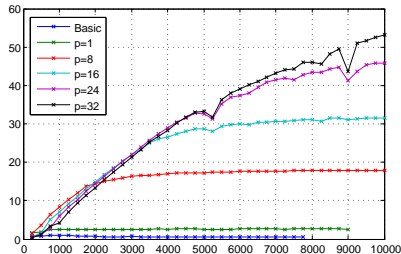
Stage 2: Performance comparison

2x Intel Xeon L5420 (8 cores)



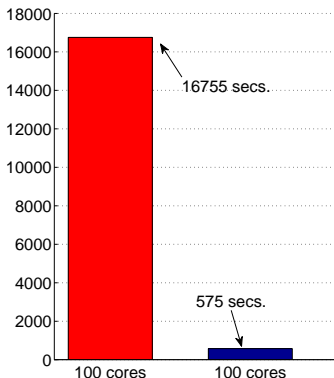
Up to 13x faster

4x AMD Opteron 6134 (32 cores)



Up to 74x faster

Topic III – sample performance results of parallel QR algorithms



Overall execution time of the QR algorithm for a $16\,000 \times 16\,000$ matrix on 25 Intel Xeon quadcore nodes (akka@HPC2N):

ScaLAPACK vs. **new implementation**.

Topic III – parallel QR and QZ multishift algorithms

- Dense eigenvalue solvers – library software
- ***Chasing multiple bulges in parallel***
- ***Aggressive early deflation in parallel***
- Overview of new implementation
- Computational experiments

Robert Granat, Bo Kågström, and Daniel Kressner, A Novel Parallel QR Algorithm for Hybrid Distributed Memory HPC Systems, *SIAM J. Sci. Comput.* 32(4):2345–2378, 2010.

Bo Kågström, Daniel Kressner, and Meiyue Shao, On Aggressive Early Deflation in Parallel Variants of the QR Algorithm, *PARA 2010, LNCS*, Springer 2011 (to appear).

Also contributions by Björn Adlerborn (parallel HT and QZ).

QR algorithm in a nutshell

- **Step 1** (*non-iterative*). Reduction to **Hessenberg** form:

$$Q_1^T A Q_1 = H = \begin{array}{|c} \triangle \\ \hline \end{array}$$

- **Step 2** (*iterative*). Reduction from Hessenberg to **Schur** form:

$$Q_2^T H Q_2 = T = \begin{array}{|c} \triangle \\ \hline \triangle \\ \hline \triangle \\ \hline \end{array}$$

Optional:

- **Step 0** **Balancing** - reduces norm of A and isolates directly deflatable eigenvalues
- **Step 3** **Eigenvalue reordering** in Schur form

Chasing multiple bulges in parallel

- Implicit QR – single bulge case
- Multiple bulges – loosely vs. tightly coupled
- Multiple chains of tightly coupled bulges

An implicit shifted QR iteration

- Assume H in **Hessenberg** form (after Step 1)
- Iteration in QR algorithm chooses, $k \ll n$ **shifts**

$$\text{sigma} = \text{eig}(H(n-k+1:n, n-k+1:n))$$

- Usually $k = 2$
- Compute **1st column** of the **shift polynomial**:

$$v = (H - \sigma_1 I)(H - \sigma_2 I)e_1 = \begin{bmatrix} X \\ X \\ X \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Bulge introduction in implicit QR iteration

- $Q_0 =$ Householder reflection with $Q_0 v = \gamma e_1$

$$H \leftarrow Q_0^T H Q_0 = \begin{bmatrix} \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \dots \\ \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \dots \\ \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \dots \\ \hat{X} & \hat{X} & \hat{X} & X & X & X & X & \dots \\ 0 & 0 & 0 & X & X & X & X & \dots \\ 0 & 0 & 0 & 0 & X & X & X & \dots \\ 0 & 0 & 0 & 0 & 0 & X & X & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

- Hessenberg structure of H disturbed by a so called **bulge**

Bulge chasing in implicit QR iteration

Chase the bulge by Householder reflections:

$$H \rightarrow \begin{bmatrix} X & \widehat{X} & \widehat{X} & \widehat{X} & X & X & X & \dots \\ \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \dots \\ \widehat{0} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \dots \\ \widehat{0} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \dots \\ 0 & \widehat{X} & \widehat{X} & \widehat{X} & X & X & X & \dots \\ 0 & 0 & 0 & 0 & X & X & X & \dots \\ 0 & 0 & 0 & 0 & 0 & X & X & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \rightarrow \begin{bmatrix} X & X & \widehat{X} & \widehat{X} & \widehat{X} & X & X & \dots \\ X & X & \widehat{X} & \widehat{X} & \widehat{X} & X & X & \dots \\ 0 & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \dots \\ 0 & \widehat{0} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \dots \\ 0 & \widehat{0} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \dots \\ 0 & 0 & \widehat{X} & \widehat{X} & \widehat{X} & X & X & \dots \\ 0 & 0 & 0 & 0 & 0 & X & X & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \rightarrow \dots$$

Tightly coupled chain of bulges

Key to efficient serial and parallel implementations: **introduce another bulge before chasing off the first one.**

$$H \rightarrow \begin{bmatrix} \hat{X} & \hat{X} & \hat{X} & X & X & X & X & \dots \\ \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \dots \\ \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \dots \\ \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \hat{X} & \dots \\ 0 & 0 & 0 & X & X & X & X & \dots \\ 0 & 0 & 0 & X & X & X & X & \dots \\ 0 & 0 & 0 & X & X & X & X & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

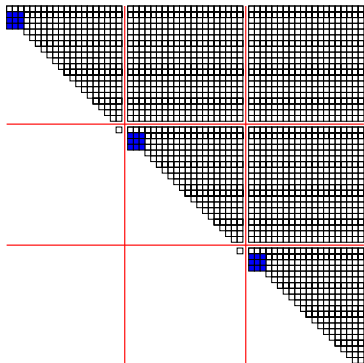
- Serial: [Braman/Byers/Mathias'02], [Lang'02].
- Parallel: [van de Geijn'93], [Watkins'94], [Henry/Watkins/Dongarra'02] and others.

Multiple chains of tightly coupled bulges

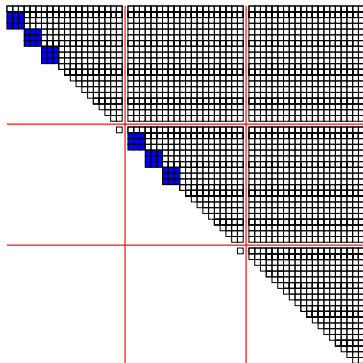
ScaLAPACK: At most one bulge per diagonal block in the process mesh. **Loosely coupled bulges.**

New implementation: Chains of **tightly coupled bulges.**

ScaLAPACK

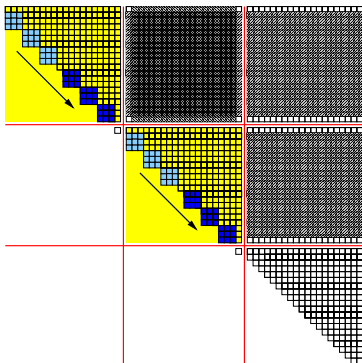


New implementation



Chasing multiple bulges in parallel – intra-block case

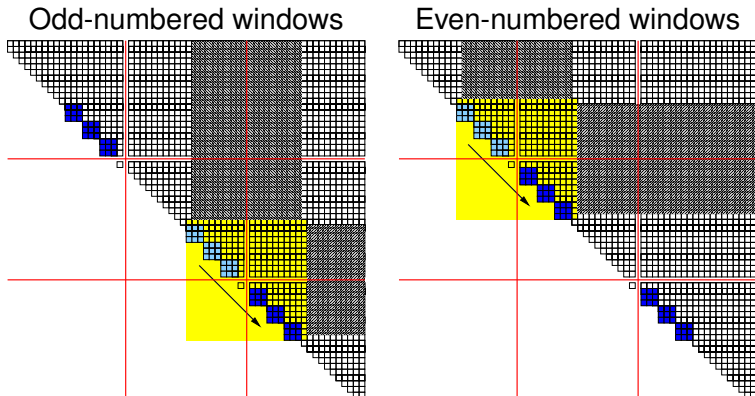
In new implementation, entire chains of bulges are chased.



An **intra-block** (local) chase of bulge chains.

Chasing multiple bulges in parallel – inter-block case

In new implementation, entire chains of bulges are chased.



An **inter-block** (cross process border) chase of bulge chains.

Summary of parallel bulge chasing

- Chains of bulges (instead of individual bulges) allow to utilize **level 3 BLAS** (delaying and accumulation of transformations).
- Large message size for cross-border communication \rightsquigarrow less penalty from latencies.
- Multithreading on each node using OpenMP directives.
- **Intro example:** decreases wall clock execution time by a factor of 5–6, in comparison to existing ScaLAPACK code (combined with multithreading on each node).

Aggressive early deflation in parallel

- Deflation strategies
- Illustration of AED
- AED in parallel

Deflation strategies

- Classical deflation criterion in **ScaLAPACK**:

$$|h_{i+1,i}| \leq \mathbf{u} \max\{|h_{i,i}|, |h_{i+1,i+1}|\},$$

with \mathbf{u} unit roundoff (double precision $\mathbf{u} \approx 1.1 \times 10^{-16}$).

- **Aggressive early deflation (AED) strategy** by Braman, Byers, and Mathias (2002) is implemented in LAPACK. Now, also in **new parallel implementation**.
- AED often detects convergence much earlier and significantly reduces average #shifts to deflate an eigenvalue.

Illustration of AED

Choose deflation window size $n_{\text{win}} \ll n$ and partition H :

$$H = \begin{matrix} & & n-n_{\text{win}}-1 & 1 & n_{\text{win}} \\ & n-n_{\text{win}}-1 & & & \\ & 1 & & & \\ & n_{\text{win}} & & & \end{matrix} \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ 0 & H_{32} & H_{33} \end{bmatrix}.$$

Compute Schur decomposition of H_{33} and update rest of H :

$$H \rightarrow \left[\begin{array}{ccc|ccccc} \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & X & X & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & X & X & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \hline \dots & 0 & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & 0 & \widehat{X} & 0 & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & 0 & \widehat{X} & 0 & 0 & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & 0 & \widehat{X} & 0 & 0 & 0 & \widehat{X} & \widehat{X} \\ \dots & 0 & \widehat{X} & 0 & 0 & 0 & 0 & \widehat{X} \end{array} \right]$$

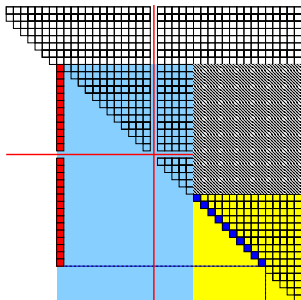
Illustration of AED

- Search for tiny trailing elements of newly introduced **spike** under **different eigenvalue orderings** of H_{33} .
- Return undeflatable part back to Hessenberg form.

For example, if 2 eigenvalues could be deflated:

$$H \rightarrow \left[\begin{array}{cc|ccccc} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & X & X & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & X & X & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \hline \dots & 0 & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & 0 & \widehat{X} & 0 & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & 0 & \widehat{X} & 0 & 0 & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & 0 & 0 & 0 & 0 & 0 & \widehat{X} & \widehat{X} \\ \dots & 0 & 0 & 0 & 0 & 0 & 0 & \widehat{X} \end{array} \right] \rightarrow \left[\begin{array}{cc|cc|cc} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & X & X & \widehat{X} & \widehat{X} & \widehat{X} & X & X \\ \dots & X & X & \widehat{X} & \widehat{X} & \widehat{X} & X & X \\ \hline \dots & 0 & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & 0 & \widehat{0} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \dots & 0 & \widehat{0} & 0 & \widehat{X} & \widehat{X} & \widehat{X} & \widehat{X} \\ \hline \dots & 0 & 0 & 0 & 0 & 0 & X & X \\ \dots & 0 & 0 & 0 & 0 & 0 & 0 & X \end{array} \right]$$

Reorganize AED s.t. groups of eigenvalues are reordered.



- Reordering performed locally in *yellow* computational window.
- Afterwards, entire group of **undeflatable eigenvalues** reordered to top left corner with a parallel algorithm described in [Granat/Kågström/Kressner'09].

Computational experiments

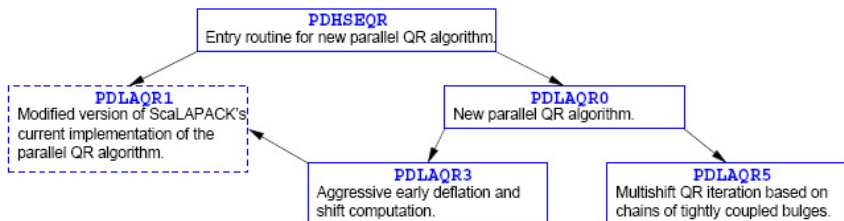
- Target parallel systems (*akka* and *sarek*)
- Radom matrices (fullrand, hessrand)
- Benchmark examples (BBM, Matrix Market, NEP collection)

$P_r \times P_c \times P_t$	Fullrand	$n =$			
	4 000	8 000	16 000	32 000	
$1 \times 1 \times 1$	9332 226	79943 1478			
$2 \times 2 \times 1$	2761 112	20161 640			
$4 \times 4 \times 1$	1174 69	8582 265	67779 1644		
$6 \times 6 \times 1$	697 60	5192 194	32920 1007	∞ 6218	
$8 \times 8 \times 1$	418 57	3671 152	20856 595	∞ 4164	
$10 \times 10 \times 1$	368 63	2589 165	16755 516	∞ 3046	
$1 \times 1 \times 4$	2592 173	18324 913			
$2 \times 2 \times 4$	1617 126	11032 591			
$3 \times 3 \times 4$	834 102	5692 408	38834 2327		
$4 \times 4 \times 4$	612 76	3986 277	25823 1332	∞ 9250	
$5 \times 5 \times 4$	474 70	2971 203	18934 1061	∞ 6568	

Execution times of **ScaLAPACK** vs. **new implementation** (in seconds).
 NIBBLE = 50; (if more than 50% evs. inside AED window are deflated, skip
 a QR sweep and start a new AED)

Up to 40x improvements!

Software structure and some recent progress



- **Performance:** Significant reduction of total execution time (adopted a second level of AED in PDLAQR1).
- **Accuracy:** Now all relative residual norms are $O(\text{machep})$ (identified and fixed several anomalies).
- **Scalability issues:** Under investigation (dynamic NIBBLE and cross over points, doing AED on a subgrid for deflation and generating new shifts).

$P_r \times P_c \times P_t$	Fullrand	$n =$							
	4 000	8 000	16 000	32 000					
$1 \times 1 \times 1$	148	135	868	934					
$2 \times 2 \times 1$	65	58	439	362					
$4 \times 4 \times 1$	34	28	169	128	1065	866			
$6 \times 6 \times 1$	40	26	116	83	591	511	3074	3236	
$8 \times 8 \times 1$	38	23	103	68	378	387	2358	1938	
$10 \times 10 \times 1$	43	24	107	69	370	333	2021	1641	
$1 \times 1 \times 4$	158	138	756	772					
$2 \times 2 \times 4$	110	86	502	370	2115	2102			
$3 \times 3 \times 4$	97	68	385	251	1579	1324			
$4 \times 4 \times 4$	74	49	254	169	1144	951	5186	4920	
$5 \times 5 \times 4$	66	42	197	129	925	662	3520	3273	

Execution times of PDHSEQR (in seconds) with **one** and two AED-levels. NIBBLE = 50.

Improvements up to 40% — on average 20% improvements!

$n = 100.000 \rightsquigarrow$ up to 50% improvements $\times 2$

- 1024 cores organized in 32×32 process grid
- Total execution time 8h 30min \rightsquigarrow 5h 05min (60%)
- Balancing = 20min
- Reduction to Hessenberg form = 1h 7min
- QR algorithm = 7h 3min \rightsquigarrow 3h 38min (52%)
 - 12 QR iterations with $\approx 2000 - 4000$ shifts
 - $n_{\text{win}} = 6145$
 - 80% of execution time spent on AED!
 - 0.44 shifts per deflated eigenvalue \rightsquigarrow 0.34
- QR algorithm: 2nd AED-level on a 8×8 subgrid \rightsquigarrow 1h 44min (48%)
 - $n_{\text{win}} = 6145$
 - 37% of execution time spent on AED!
 - \rightsquigarrow 0.35 shifts per deflated eigenvalue

Topic III - overall summary

- New parallel implementation significantly outperforms existing ScaLAPACK implementation
- Parallel QR iterations by chasing several bulge chains
- Parallel AED by algorithmic reformulation(s)
- Tremendous implementation effort

- B. Kågström, D. Kressner, and M. Shao, [On Aggressive Early Deflation in Parallel Variants of the QR Algorithm](#), *PARA 2010, LNCS*, Springer 2011 (to appear).
- R. Granat, B. Kågström, and D. Kressner. [A Novel Parallel QR Algorithm for Hybrid Distributed Memory HPC Systems](#), *SIAM J. Sci. Comput.* 32(4):2345–2378, 2010.
- R. Granat, B. Kågström, and D. Kressner. [Parallel eigenvalue reordering in real Schur forms](#). *Concurrency and Computation: Practice and Experience*, 21(9):1225–1250, 2009.
- B. Adlerborn, D. Kressner, and B. Kågström. [Parallel variants of the multishift QZ algorithm with advanced deflation techniques](#). In B. Kågström et al., editor, *Applied Parallel Computing - State of the Art in Scientific Computing (PARA'06)*, LNCS Vol. 4699, pages 117–126, Springer, 2007.

Acknowledgements

- All co-workers and the Umeå research group
- This research was conducted using the resources of the [High Performance Computing Center North \(HPC2N\)](#).
- Financial support has been provided by the *Swedish Research Council* under grant VR 70625701 and by the *Swedish Foundation for Strategic Research* under grant A3 02:128.



Vetenskapsrådet



SWEDISH FOUNDATION *for*
STRATEGIC RESEARCH