Scalable block methods and preconditioning for hardware efficient sparse eigensolutions

Achim Basermann, Melven Röhrig-Zöllner, Jonas Thies

Simulation and Software Technology
German Aerospace Center, Cologne

DLR

Background

Aim: Calculate a set of extreme eigenpairs (λ_i, v_i) of a large sparse matrix:

$$Av_i = \lambda_i v_i$$

Project: Equipping Sparse Solvers for Exascale (ESSEX) of the DFG SPPEXA programme

DLR

Outline

DLR

Outline

DLR

Jacobi-Davidson QR method (Fokkema, 1998)

Sketch of the algorithm

1: **while** not converged **do** ▷ Outer iteration
2: Project the problem to a small subspace
3: Solve the small eigenvalue problem
4: Calculate an approximation and its residual
5: Approximately solve the correction equation ▷ Inner iteration
6: Orthogonalize the new direction
7: Enlarge the subspace
8: **end while**

Block JDQR method

Idea

- Calculate corrections for n_b eigenvalues at once

Numerical properties

Block JDQR method

Idea

- Calculate corrections for n_b eigenvalues at once
- Block correction equation with $\tilde{Q} = \begin{pmatrix} Q & \tilde{v}_1 & \dots & \tilde{v}_{n_b} \end{pmatrix}$:

$$(I - \tilde{Q}\tilde{Q}^T)(A - \tilde{\lambda}_i I)(I - \tilde{Q}\tilde{Q}^T)w_{k+i} = -r_i \qquad i = 1, \dots, n_b$$

Numerical properties

Block JDQR method

Idea

- Calculate corrections for n_b eigenvalues at once
- Block correction equation with $\tilde{Q} = \begin{pmatrix} Q & \tilde{v}_1 & \dots & \tilde{v}_{n_b} \end{pmatrix}$:

$$(I - \tilde{Q}\tilde{Q}^T)(A - \tilde{\lambda}_i I)(I - \tilde{Q}\tilde{Q}^T)w_{k+i} = -r_i \qquad i = 1, \dots, n_b$$

$\rightarrow$ Approximately solve n_b linear systems at once

Numerical properties

Block JDQR method

Idea

- Calculate corrections for n_b eigenvalues at once
- Block correction equation with $\tilde{Q} = \begin{pmatrix} Q & \tilde{v}_1 & \dots & \tilde{v}_{n_b} \end{pmatrix}$:

$$(I - \tilde{Q}\tilde{Q}^T)(A - \tilde{\lambda}_i I)(I - \tilde{Q}\tilde{Q}^T)w_{k+i} = -r_i \qquad i = 1, \dots, n_b$$

$\rightarrow$ Approximately solve n_b linear systems at once

- Provides new directions $w_{k+1}, \dots, w_{k+n_b}$ for the subspace iteration

Numerical properties

Block JDQR method

Idea

- Calculate corrections for n_b eigenvalues at once
- Block correction equation with $\tilde{Q} = \begin{pmatrix} Q & \tilde{v}_1 & \dots & \tilde{v}_{n_b} \end{pmatrix}$:

$$(I - \tilde{Q}\tilde{Q}^T)(A - \tilde{\lambda}_i I)(I - \tilde{Q}\tilde{Q}^T)w_{k+i} = -r_i \qquad i = 1, \dots, n_b$$

$\rightarrow$ Approximately solve n_b linear systems at once

- Provides new directions $w_{k+1}, \dots, w_{k+n_b}$ for the subspace iteration

Numerical properties

- More robust
- Usually needs more operations

Block JDQR method

Sketch of the complete algorithm

1:
2: **while** not converged **do**
3: Project the problem to a small subspace
4: Solve the small eigenvalue problem
5: Calculate an n_b approximations and their residual
6:
7:
8: Approximately solve the n_b correction equations
9: Block-Orthogonalize the new directions (using TSQR)
10: Enlarge the subspace
11: **end while**

Block JDQR method

Sketch of the complete algorithm

1: Setup initial subspace
2: **while** not converged **do**
3: Project the problem to a small subspace
4: Solve the small eigenvalue problem
5: Calculate an n_b approximations and their residual
6: Lock converged eigenvalues
7: Shrink subspace if required (thick restart)
8: Approximately solve the n_b correction equations
9: Block-Orthogonalize the new directions (using TSQR)
10: Enlarge the subspace
11: **end while**

Outline

Required linear algebra operations

Sparse matrix-multiple-vector multiplication (spMMVM)

Block vector operations

Required linear algebra operations

Sparse matrix-multiple-vector multiplication (spMMVM)

- Large distributed sparse matrix A in SELL-C-σ format
- Distributed blocks of vectors $X, Y \in \mathbb{R}^{n \times n_b}$
- Shifted spMMVM: $y_i \leftarrow (A - \tilde{\lambda}_i I)x_i, \quad i = 1, \ldots, n_b$

Block vector operations

Required linear algebra operations

Sparse matrix-multiple-vector multiplication (spMMVM)

- Large distributed sparse matrix A in SELL-C-σ format
- Distributed blocks of vectors $X, Y \in \mathbb{R}^{n \times n_b}$
- Shifted spMMVM: $y_i \leftarrow (A - \tilde{\lambda}_i I)x_i, \quad i = 1, \ldots, n_b$

Block vector operations

- Different types of operations:

	local	all-reduction
BLAS 1	$Y \leftarrow X + Y$	$\|x_i\|, i = 1, \ldots, n_b$
BLAS 3	$Y \leftarrow XM$	$M \leftarrow X^T Y$

- Redundantly stored small matrices $M \in \mathbb{R}^{n_b \times n_b}$

Block vector operations

Background

- All operations are memory bound.
  (also the GEMM, as all matrices are very tall and skinny)
- The code balance accurately predicts the node-level performance!

Results of blocking

Block vector operations

Background

- All operations are memory bound.
  (also the GEMM, as all matrices are very tall and skinny)
- The code balance accurately predicts the node-level performance!

Results of blocking

- Faster BLAS 3 operations (e.g. $Y \leftarrow XM$)
- Message aggregation for all-reductions (e.g. $M \leftarrow X^T Y$)

$\rightarrow$ Improved performance of some operations

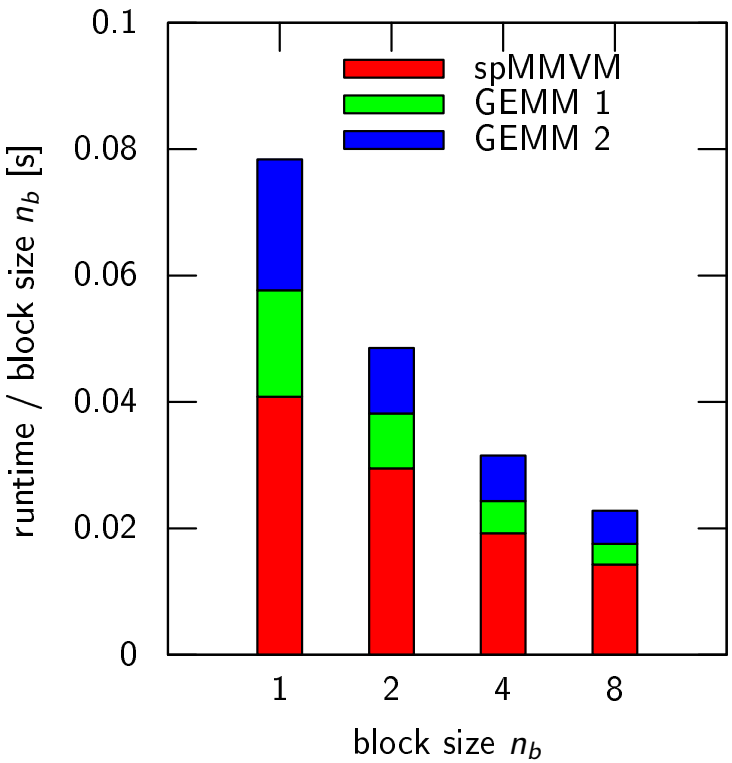
Complete Jacobi-Davidson Operator

Setup

- spMMVM + 2× GEMM:

$$y_i \leftarrow (I - QQ^T)(A - \tilde{\lambda}_i I)x_i$$

  with $Q \in \mathbb{R}^{n \times 8}$, $i = 1, \ldots, n_b$
- Matrix with $n \approx 10^7$, $n_{nzr} \approx 15$
- SELL-C-σ format
- 10-core Intel Ivy Bridge CPU



Outline

Frameworks from the ESSEX project

phist (Pipelined Hybrid-parallel Linear Solver Toolkit)

GHOST (General Hybrid Optimized Sparse Toolkit)

Frameworks from the ESSEX project

phist (Pipelined Hybrid-parallel Linear Solver Toolkit)

- General C-interface to linear algebra libraries:
  - GHOST
  - Trilinos (C++, `http://trilinos.sandia.gov`)
  - *builtin* (for prototyping, row-major storage, Fortran + C99)

GHOST (General Hybrid Optimized Sparse Toolkit)

Frameworks from the ESSEX project

phist (Pipelined Hybrid-parallel Linear Solver Toolkit)

- General C-interface to linear algebra libraries:
  - GHOST
  - Trilinos (C++, `http://trilinos.sandia.gov`)
  - *builtin* (for prototyping, row-major storage, Fortran + C99)
- Iterative solvers

GHOST (General Hybrid Optimized Sparse Toolkit)

Frameworks from the ESSEX project

phist (Pipelined Hybrid-parallel Linear Solver Toolkit)

- General C-interface to linear algebra libraries:
  - GHOST
  - Trilinos (C++, `http://trilinos.sandia.gov`)
  - *builtin* (for prototyping, row-major storage, Fortran + C99)
- Iterative solvers
- Large test framework

GHOST (General Hybrid Optimized Sparse Toolkit)

Frameworks from the ESSEX project

phist (Pipelined Hybrid-parallel Linear Solver Toolkit)

- General C-interface to linear algebra libraries:
  - GHOST
  - Trilinos (C++, `http://trilinos.sandia.gov`)
  - *builtin* (for prototyping, row-major storage, Fortran + C99)
- Iterative solvers
- Large test framework

GHOST (General Hybrid Optimized Sparse Toolkit)

- Developed at the RRZE in Erlangen
- Hybrid-parallel MPI+OpenMP+CUDA
- SELL-C-σ matrix format

Block pipelined GMRES or MINRES *preconditioning*

Idea

- Solve n_b linear systems $Ax_i = b_i$ at once

Block pipelined GMRES or MINRES *preconditioning*

Idea

- Solve n_b linear systems $Ax_i = b_i$ at once
- Remove converged systems and add new ones (pipelining)

Block pipelined GMRES or MINRES *preconditioning*

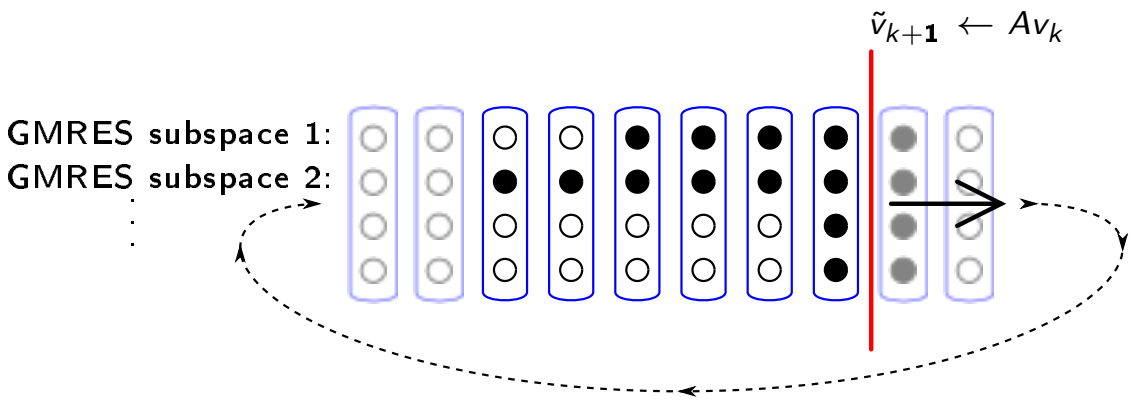
Idea

- Solve n_b linear systems $Ax_i = b_i$ at once
- Remove converged systems and add new ones (pipelining)
- Standard GMRES or MINRES method (for each system)

Block pipelined GMRES or MINRES *preconditioning*

Idea

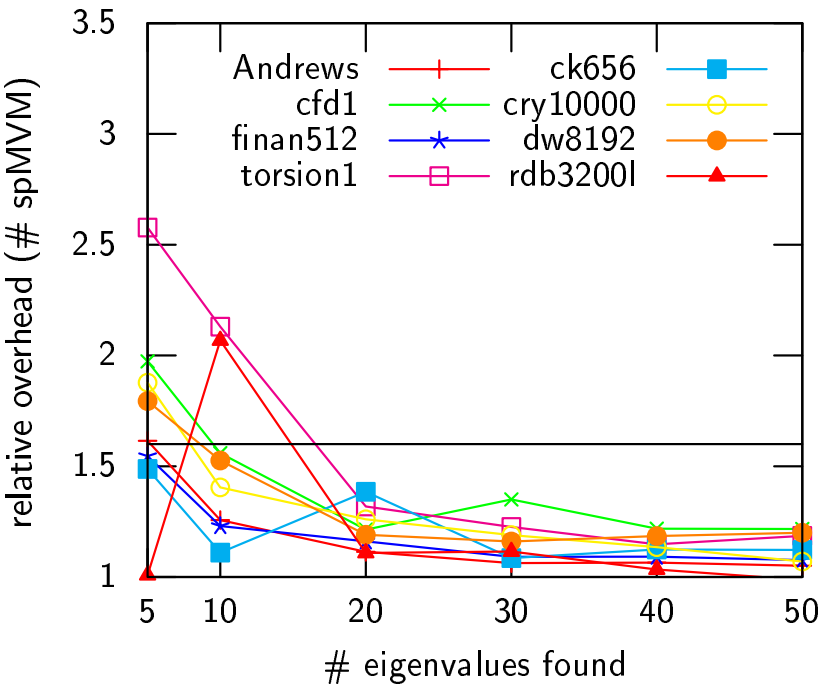
- Solve n_b linear systems $Ax_i = b_i$ at once
- Remove converged systems and add new ones (pipelining)
- Standard GMRES or MINRES method (for each system)



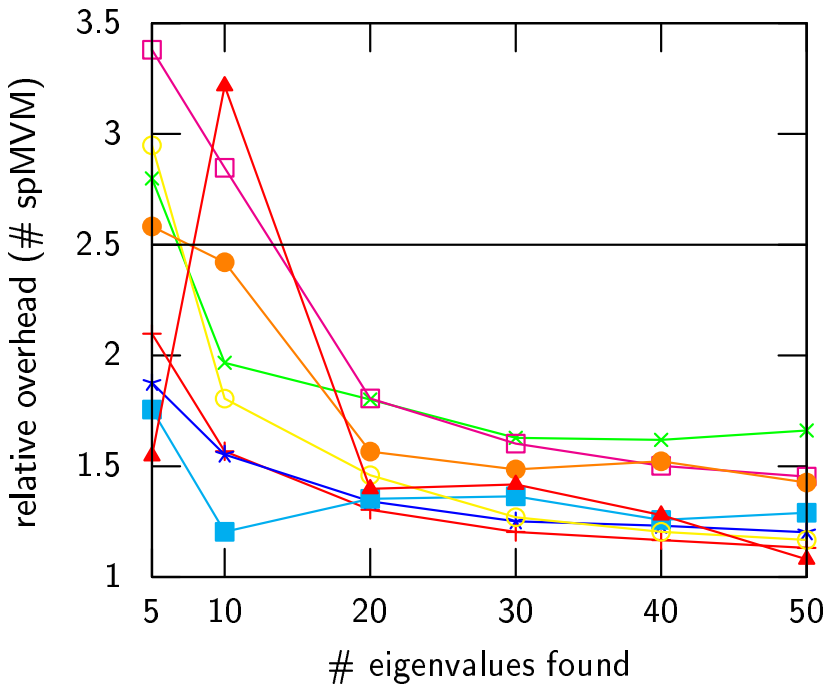
Outline

Numerical behavior

Block size 2



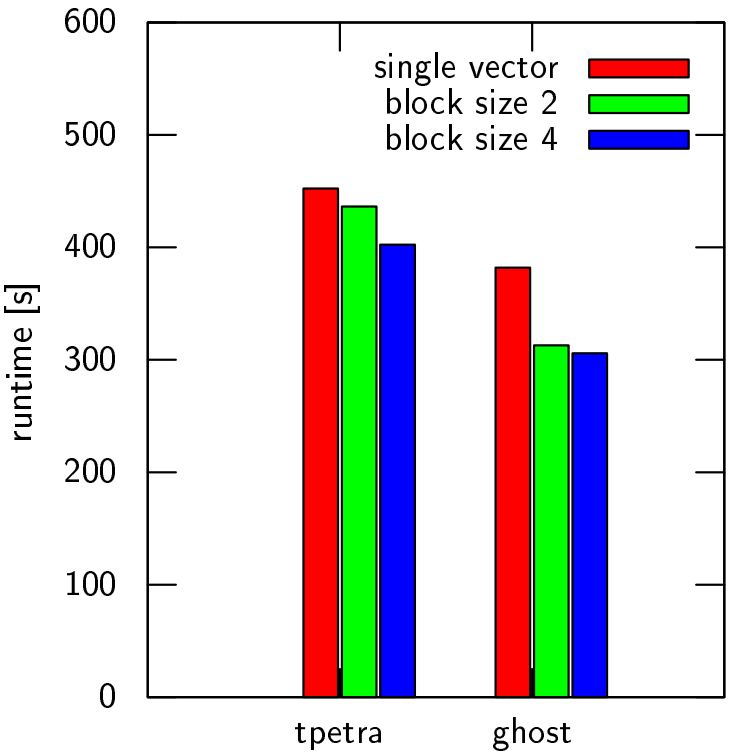
Block size 4



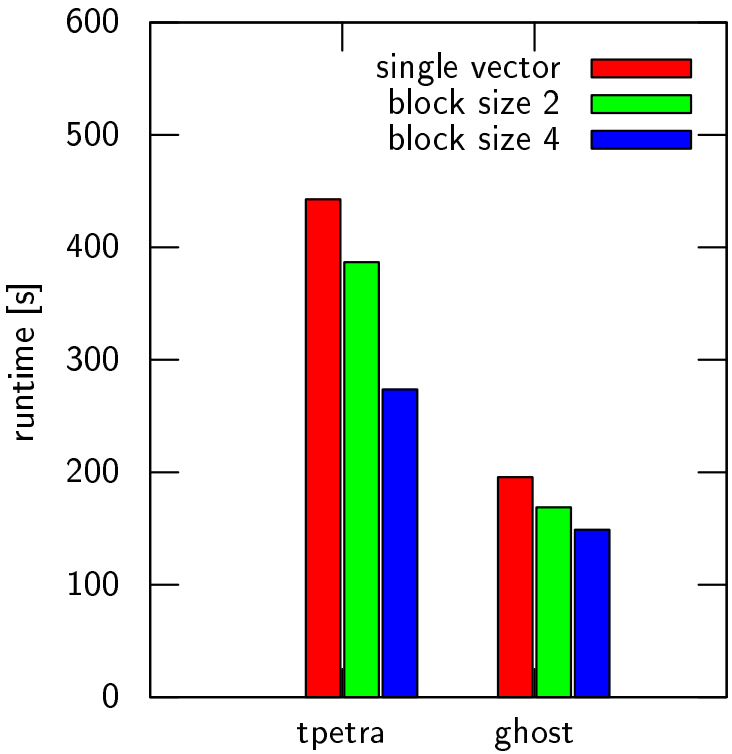
Performance of the complete algorithm (0)

Seeking 20 eigenvalues of spinSZ26 ($1 \cdot 10^7$ rows, $1.5 \cdot 10^8$ nonzeros)

Single socket (10 cores)



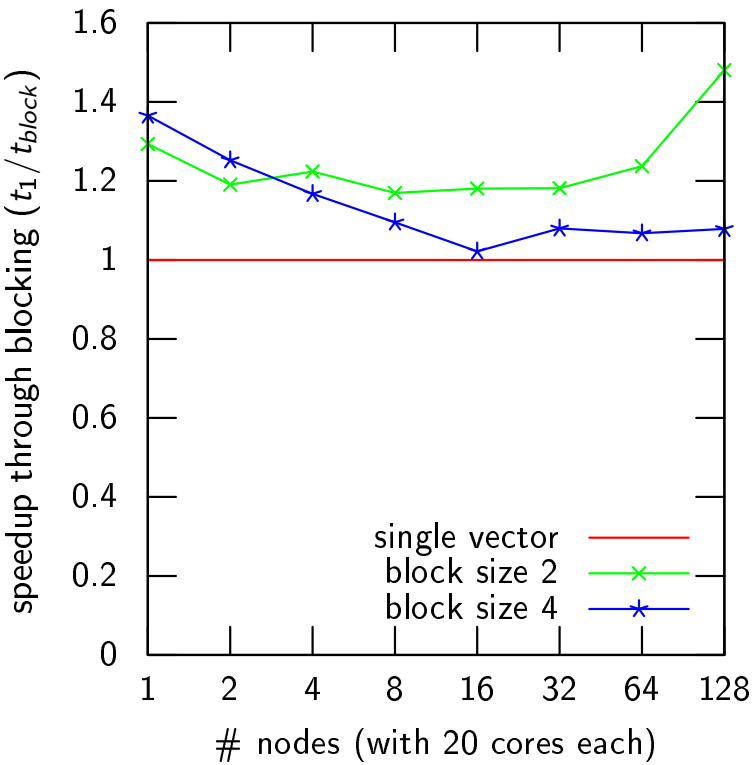
Complete node (2×10 cores)



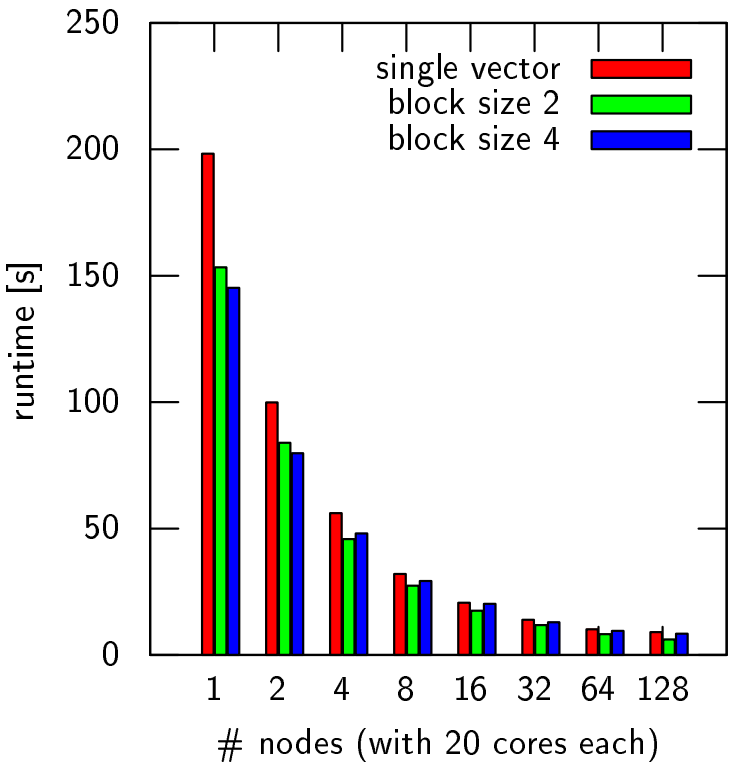
Performance of the complete algorithm (1)

Seeking 20 eigenvalues of spinSZ26 ($1.0 \cdot 10^7$ rows, $1.5 \cdot 10^8$ nonzeros)

Block speedup



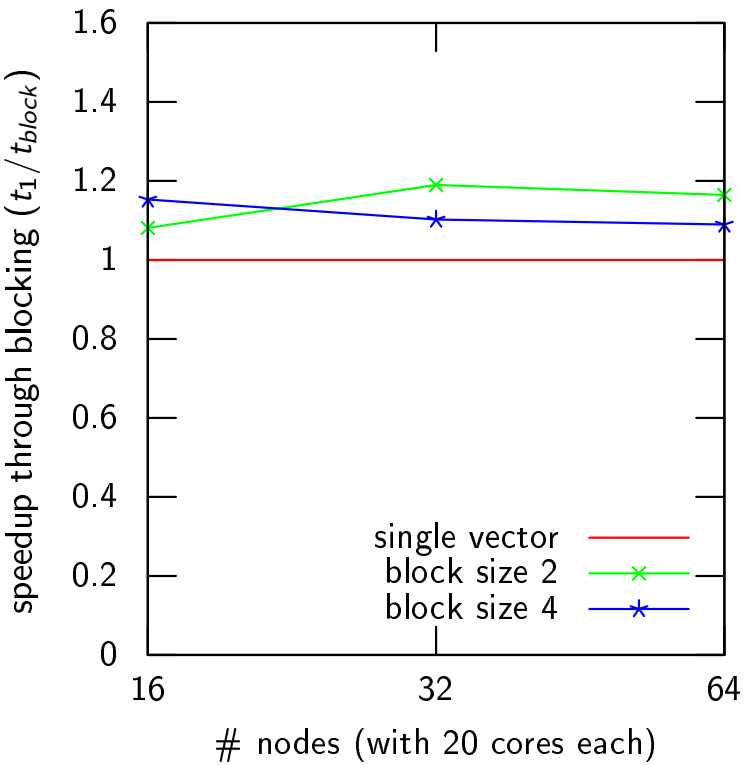
Strong scaling



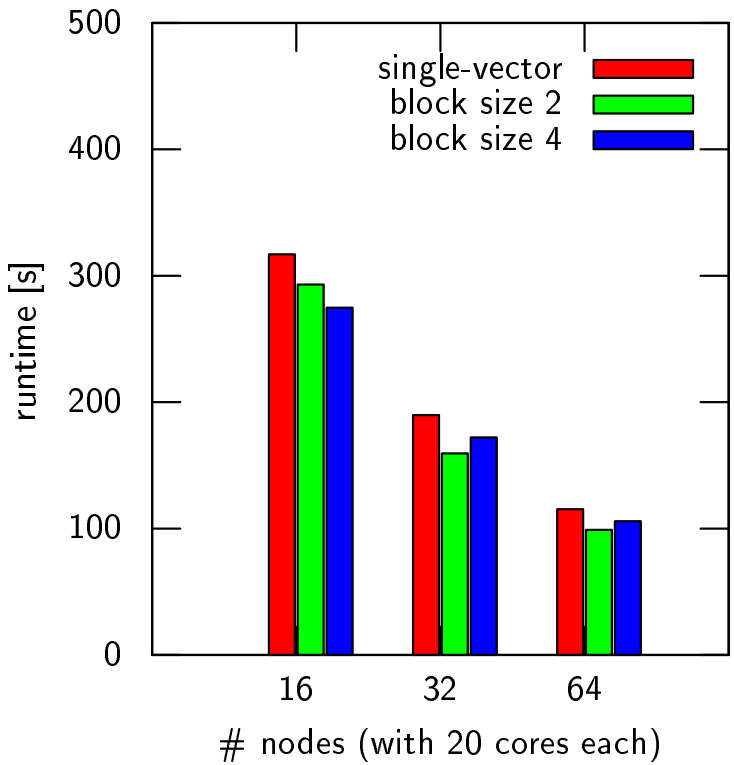
Performance of the complete algorithm (2)

Seeking 20 eigenvalues of spinSZ30 ($1.6 \cdot 10^8$ rows, $2.6 \cdot 10^9$ nonzeros)

Block speedup



Strong scaling



Conclusion (1)

Block JDQR algorithm

- Performance analysis of block operations:
  - Impact of memory layout (row- instead of col.-major)
  - Node-level: better code balance
  - Inter-node: message aggregation

Conclusion (1)

Block JDQR algorithm

- Performance analysis of block operations:
  - Impact of memory layout (row- instead of col.-major)
  - Node-level: better code balance
  - Inter-node: message aggregation
- Numerical behavior
  - Slight increase of operations
  - Overhead small for > 10 eigenvalues

Conclusion (1)

Block JDQR algorithm

- Performance analysis of block operations:
  - Impact of memory layout (row- instead of col.-major)
  - Node-level: better code balance
  - Inter-node: message aggregation
- Numerical behavior
  - Slight increase of operations
  - Overhead small for > 10 eigenvalues

$\rightarrow$ Improved performance by factor 1.2-1.4

Conclusion (2)

Outlook

- Numerical improvements:
  - Parallel preconditioning of the linear problems

Conclusion (2)

Outlook

- Numerical improvements:
  - Parallel preconditioning of the linear problems
- Algorithmic improvements:
  - Hiding spMMVM communication behind other operations
  - More asynchronous communication (all-reductions)

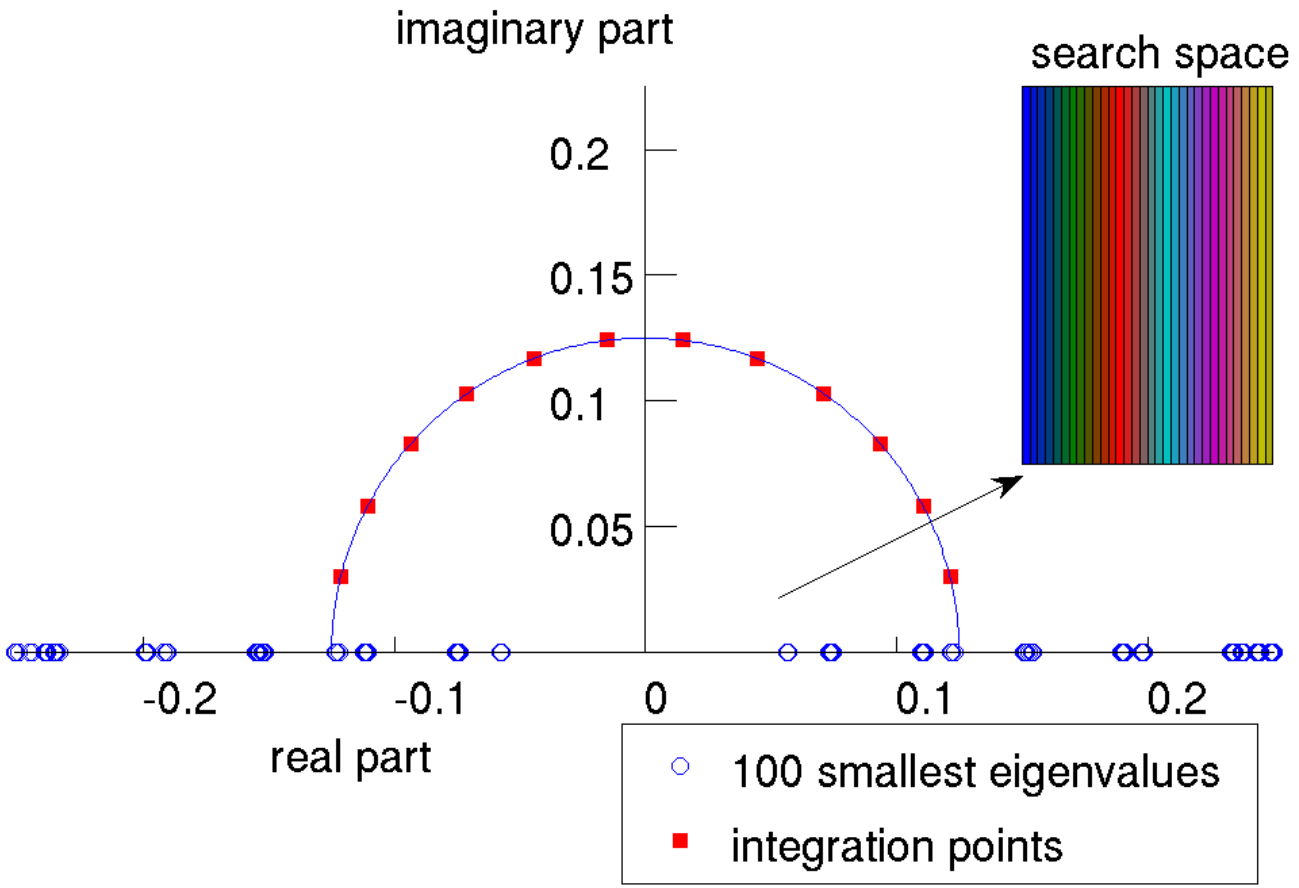
Conclusion (2)

Outlook

- Numerical improvements:
  - Parallel preconditioning of the linear problems
- Algorithmic improvements:
  - Hiding spMMVM communication behind other operations
  - More asynchronous communication (all-reductions)
- Hybrid calculations with GPUs

Outline

FEAST eigensolver (Polizzi '09)



Linear systems for FEAST/graphene

Tough:

- very large ($N \sim 10^9$ carbon atoms)
- complex symmetric and completely indefinite
- small random numbers on and around the diagonal
- spectrum essentially continuous
- shifts get very close to the spectrum

But also nice in some ways:

- 2D mesh, very sparse (~ 10 entries/row)
- many RHS/shift (block methods, etc.)

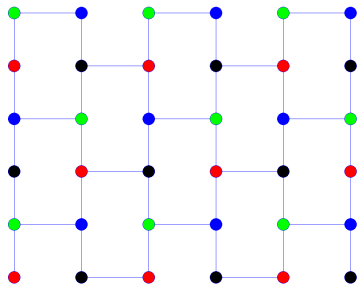
State of the art: direct solver, feasible up to a few million C atoms

The CGMN linear solver

- Björck and Elfving, 1979
- CG on the semi-definite problem $(I - Q_{SSOR})x = b$, where $Q_{SSOR} = Q_1 Q_2 \dots Q_N Q_{N-1} \dots Q_1$ is the SSOR iteration on $AA^T y = b$
- $Q_i v = (I - \omega \frac{a_i^T v}{a_i a_i^T}) a_i^T$: project v onto a_i (row i of A)
- extremely robust: A may be singular, non-square etc.
- squaring A remedies small diagonal entries
- row scaling alleviates issue of 'squared condition number'

Two parallelization strategies

Algebraic Multi-Coloring



Distance-2 coloring resolves data dependency

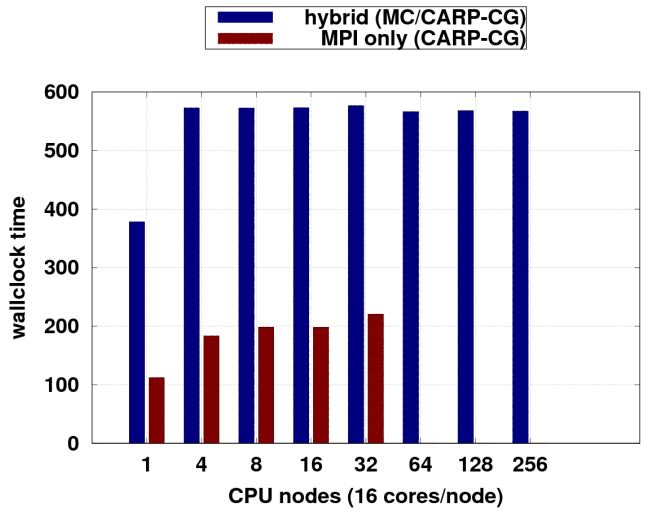
- yields fine grained parallelism (e.g. GPGPU)

CARP: Component-Averaged Row Projection (Gordon & Gordon, 2005)

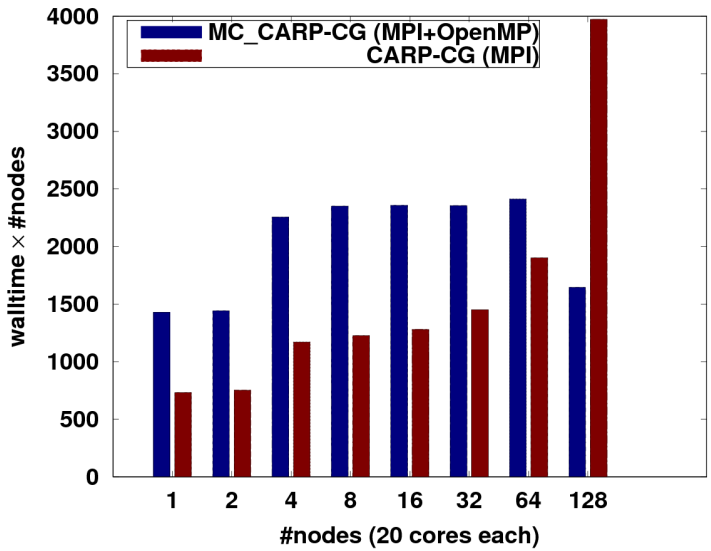
- sequential sweeps on subdomains
- exchange and average halo elements
- retains convergence properties of sequential algorithm

Idea: node-local MC with MPI-based CARP between the nodes

Weak and strong scaling up to 5k Cores



Graphene, 20M atoms/node (up to 5B)

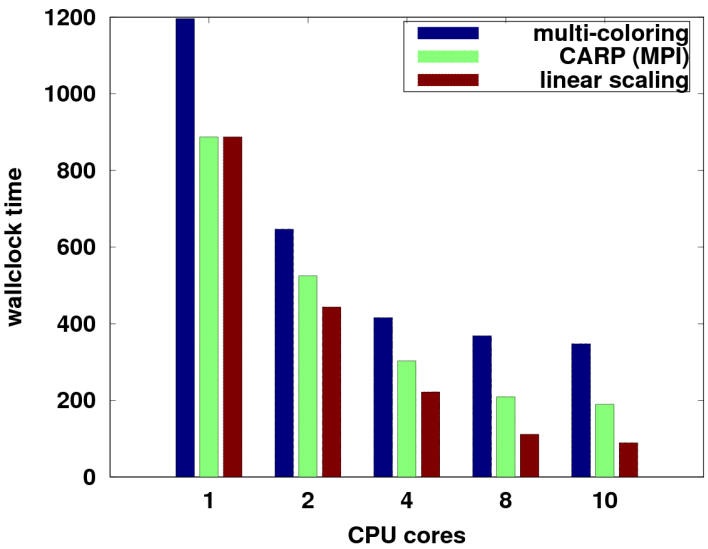


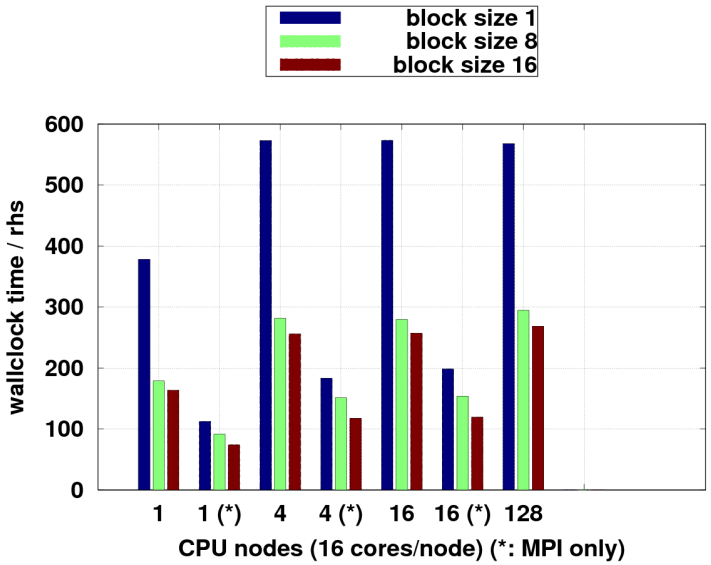
Strong scaling, 20M unknowns in total

- Coloring costs performance;
- but is more memory efficient;

- yields better strong scaling;
- and has better potential for GPUs and Xeon Phi.

Socket scaling and block Performance





- Performance penalty of coloring already at core (25%) and socket (50%) levels
- Blocking alleviates poor memory access patterns with coloring,
- PHIST/GHOST: MC_CARP kernel only available in CRS

Conclusion

FEAST for interior eigenvalues

- Robust iterative solution of the arising nearly singular equation systems with CGMN

Conclusion

FEAST for interior eigenvalues

- Robust iterative solution of the arising nearly singular equation systems with CGMN
- Scalable parallelization of CGMN with algebraic multi-coloring and CARP

Conclusion

FEAST for interior eigenvalues

- Robust iterative solution of the arising nearly singular equation systems with CGMN
- Scalable parallelization of CGMN with algebraic multi-coloring and CARP
- Blocking distinctly improves performance.