Parallel solution of large sparse eigenproblems using a Block-Jacobi-Davidson method

Melven Röhrig-Zöllner
Simulation and Software Technology

DLR

Background

Aim: Calculate a set of outer 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

Jacobi-Davidson QR method

Background

- Introduced 1998 by Fokkema et. al.
- Iteratively calculates a small set of eigenvalues (near a specific target)

Jacobi-Davidson QR method

Background

- Introduced 1998 by Fokkema et. al.
- Iteratively calculates a small set of eigenvalues (near a specific target)
- Based on a subspace iteration and a correction equation

Jacobi-Davidson QR method

Background

- Introduced 1998 by Fokkema et. al.
- Iteratively calculates a small set of eigenvalues (near a specific target)
- Based on a subspace iteration and a correction equation
- Motivated by the Rayleigh quotient iteration (RQI)

$$\text{RQI:} \begin{cases} \bar{v}_{k+1} &= (A - \lambda_k I)^{-1} v_k, \quad v_{k+1} = \frac{\bar{v}_{k+1}}{\|v_{k+1}\|} \\ \lambda_{k+1} &= v_{k+1}^T A v_{k+1} \end{cases}$$

- Can also be derived from an inexact Newton process

Jacobi-Davidson QR method

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**

Outer iteration

Subspace iteration

Deflation

Outer iteration

Subspace iteration

- *Galerkin* projection onto a small subspace $\mathcal{W} = \text{span}\{w_1, w_2, \dots\}$:

$$\begin{aligned} & Av - \lambda v \perp \mathcal{W}, && v \in \mathcal{W} \\ \Leftrightarrow \quad & (W^T A W)s - \tilde{\lambda} s = 0, && \text{for orth. basis } W \end{aligned}$$

Deflation

Outer iteration

Subspace iteration

- *Galerkin* projection onto a small subspace $\mathcal{W} = \text{span}\{w_1, w_2, \dots\}$:

$$\begin{aligned} & Av - \lambda v \perp \mathcal{W}, && v \in \mathcal{W} \\ \Leftrightarrow \quad & (W^T A W)s - \tilde{\lambda} s = 0, && \text{for orth. basis } W \end{aligned}$$

- Approximation $\tilde{\lambda}$ with $\tilde{v} = Ws$

Deflation

Outer iteration

Subspace iteration

- *Galerkin* projection onto a small subspace $\mathcal{W} = \text{span}\{w_1, w_2, \dots\}$:

$$\begin{aligned} & Av - \lambda v \perp \mathcal{W}, && v \in \mathcal{W} \\ \Leftrightarrow \quad & (W^T A W)s - \tilde{\lambda} s = 0, && \text{for orth. basis } W \end{aligned}$$

- Approximation $\tilde{\lambda}$ with $\tilde{v} = Ws$

Deflation

- Project out the already known eigenvector space $\mathcal{Q} = \text{span}\{q_1, q_2, \dots\}$:

$$\bar{A} := (I - QQ^T)A(I - QQ^T)$$

Inner iteration

Jacobi-Davidson correction equation

- Based on the current approximation $A\tilde{v} - \tilde{\lambda}\tilde{v} = r$

Inner iteration

Jacobi-Davidson correction equation

- Based on the current approximation $A\tilde{v} - \tilde{\lambda}\tilde{v} = r$
- Avoid the (near) singularity of $(A - \tilde{\lambda}I)^{-1}$ through a deflation of $\tilde{Q} = \begin{pmatrix} Q & \tilde{v} \end{pmatrix}$

Inner iteration

Jacobi-Davidson correction equation

- Based on the current approximation $A\tilde{v} - \tilde{\lambda}\tilde{v} = r$
- Avoid the (near) singularity of $(A - \tilde{\lambda}I)^{-1}$ through a deflation of $\tilde{Q} = \begin{pmatrix} Q & \tilde{v} \end{pmatrix}$

$\rightarrow$ Solve approximately:

$$(I - \tilde{Q}\tilde{Q}^T)(A - \tilde{\lambda}I)(I - \tilde{Q}\tilde{Q}^T)w_{k+1} = -r$$

Inner iteration

Jacobi-Davidson correction equation

- Based on the current approximation $A\tilde{v} - \tilde{\lambda}\tilde{v} = r$
- Avoid the (near) singularity of $(A - \tilde{\lambda}I)^{-1}$ through a deflation of $\tilde{Q} = \begin{pmatrix} Q & \tilde{v} \end{pmatrix}$

$\rightarrow$ Solve approximately:

$$(I - \tilde{Q}\tilde{Q}^T)(A - \tilde{\lambda}I)(I - \tilde{Q}\tilde{Q}^T)w_{k+1} = -r$$

- Use some steps of an iterative solver (GMRES, BiCGStab, ...)

Inner iteration

Jacobi-Davidson correction equation

- Based on the current approximation $A\tilde{v} - \tilde{\lambda}\tilde{v} = r$
- Avoid the (near) singularity of $(A - \tilde{\lambda}I)^{-1}$ through a deflation of $\tilde{Q} = \begin{pmatrix} Q & \tilde{v} \end{pmatrix}$

$\rightarrow$ Solve approximately:

$$(I - \tilde{Q}\tilde{Q}^T)(A - \tilde{\lambda}I)(I - \tilde{Q}\tilde{Q}^T)w_{k+1} = -r$$

- Use some steps of an iterative solver (GMRES, BiCGStab, ...)
- Provides a new direction w_{k+1} for the subspace iteration

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
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
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 CRS or 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 CRS or 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 $M \in \mathbb{R}^{n_b \times n_b}$

spMMVM single node

Roofline performance model

- Possible performance:

$$P_{ideal} = \min\left(P_{peak}, \frac{b_{data}}{B_{code}}\right)$$

spMMVM single node

Roofline performance model

- Possible performance:

$$P_{ideal} = \min\left(P_{peak}, \frac{b_{data}}{B_{code}}\right)$$

- Ideal code balance:

$$B_{ideal} = \frac{6}{n_b} + \frac{8}{n_{nzr}} \left[\frac{bytes}{flop}\right]$$

spMMVM single node

Roofline performance model

- Possible performance:

$$P_{ideal} = \min\left(P_{peak}, \frac{b_{data}}{B_{code}}\right)$$

- Ideal code balance:

$$B_{ideal} = \frac{6}{n_b} + \frac{8}{n_{nzr}} \left[\frac{bytes}{flop}\right]$$

$\rightarrow$ memory bounded on all current architectures

spMMVM single node

Roofline performance model

- Possible performance:

$$P_{ideal} = \min\left(P_{peak}, \frac{b_{data}}{B_{code}}\right)$$

- Ideal code balance:

$$B_{ideal} = \frac{6}{n_b} + \frac{8}{n_{nzr}} \left[\frac{bytes}{flop}\right]$$

$\rightarrow$ memory bounded on all current architectures

- Irregular accesses to X lead to higher data volume:

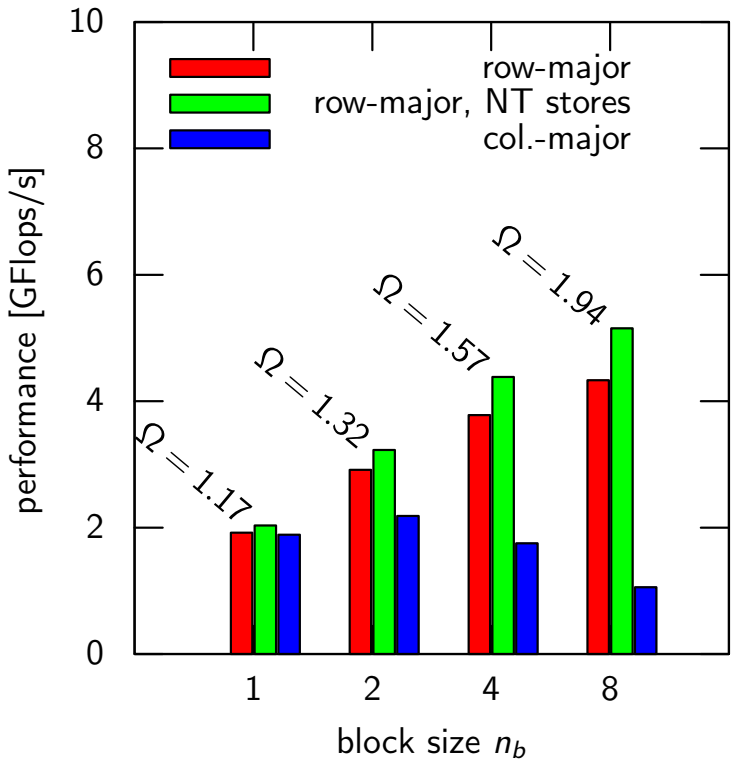
$$\Omega = \frac{V_{measured}}{V_{ideal}} \geq 1$$

spMMVM single node

Row- vs. column-major storage

Setup

- Matrix dimensions: $n \approx 10^6$ with $n_{nzr} \approx 10$ non-zeros per row
- CRS format
- 6-core Intel Westmere CPU



spMMVM single node

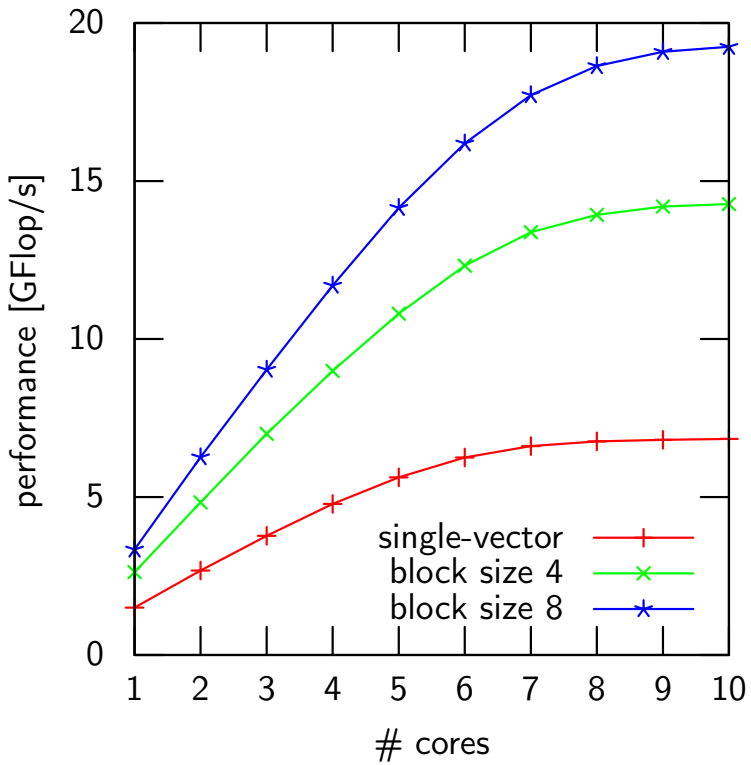
Intra-socket scaling

Setup

- Matrix dimensions: $n \approx 10^7$ with $n_{nzr} \approx 15$ non-zeros per row
- SELL-C-σ format
- 10-core Intel Ivy Bridge CPU

Results

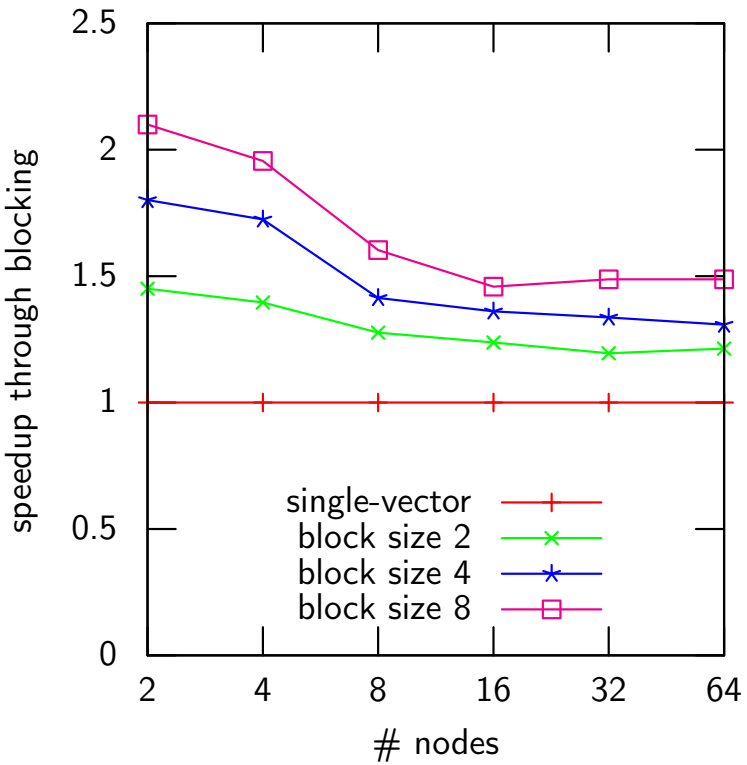
block size	1	4	8
Ω	1.08	1.54	1.97
Speedup	1.0	2.1	2.8



spMMVM inter-node

Setup

- Strong scaling:
  $n \approx 4 \cdot 10^7$, $n_{nzr} \approx 15$
- Distributed CRS
- RRZE's Emmy-cluster, each node:
  2× 10-core Intel Ivy Bridge
- Timings from completed algorithm



Explanation

Two counteracting effects:

- Communication volume increases independent of blocking
- Message aggregation

Block vector operations

Background

- All operations are memory bounded.
  (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 bounded.
  (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

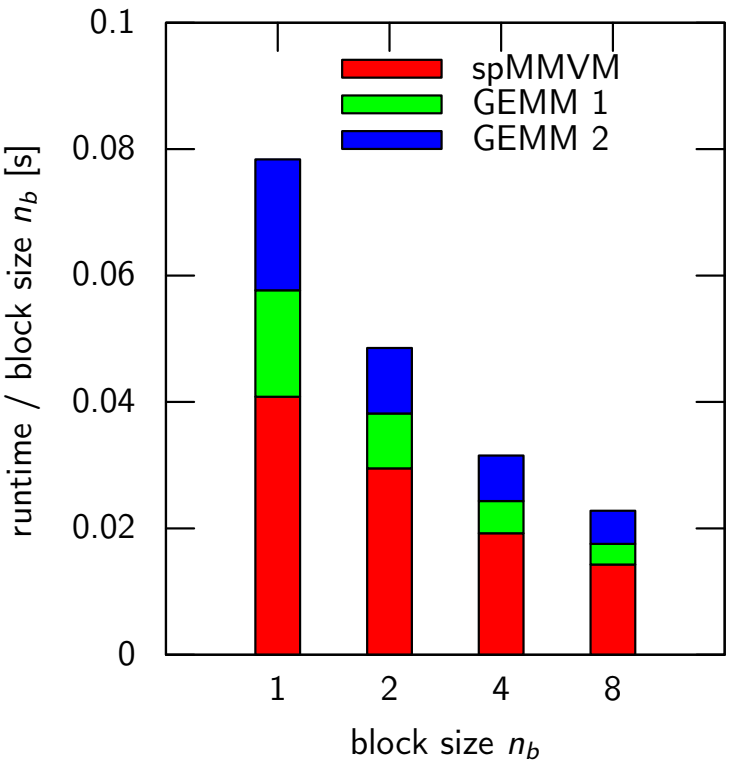
Complete Jacobi-Davidson Operator

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 algorithm

Idea

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

Block pipelined GMRES algorithm

Idea

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

Block pipelined GMRES algorithm

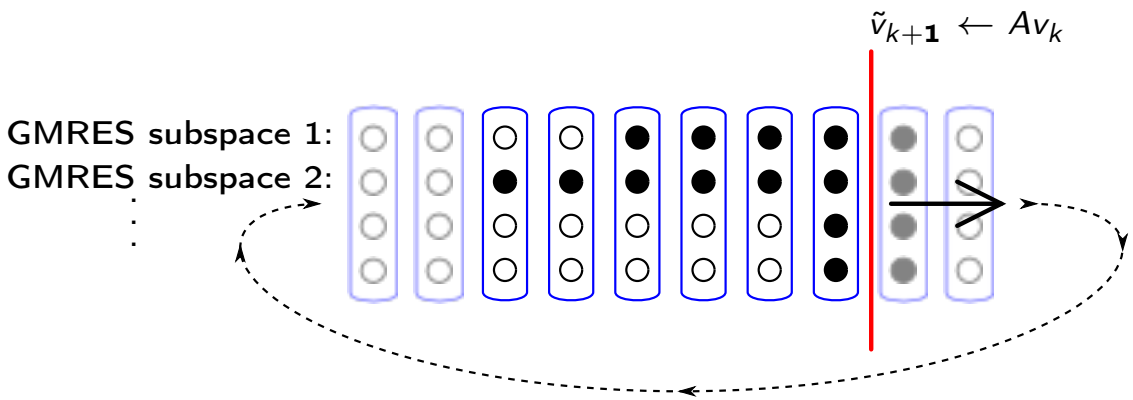
Idea

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

Block pipelined GMRES algorithm

Idea

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



Kernel routines

Performance pitfalls

- Common BLAS libraries are slow for the block vector GEMM operations here (e.g. $M \leftarrow X^T Y$).

Implementation details

Kernel routines

Performance pitfalls

- Common BLAS libraries are slow for the block vector GEMM operations here (e.g. $M \leftarrow X^T Y$).
- Getting NUMA accesses right is difficult in a complex library. (Works fine in GHOST!)

Implementation details

Kernel routines

Performance pitfalls

- Common BLAS libraries are slow for the block vector GEMM operations here (e.g. $M \leftarrow X^T Y$).
- Getting NUMA accesses right is difficult in a complex library. (Works fine in GHOST!)
- Strided accesses in row-major block vectors are slow. (Hidden by the interface.)

Implementation details

Kernel routines

Performance pitfalls

- Common BLAS libraries are slow for the block vector GEMM operations here (e.g. $M \leftarrow X^T Y$).
- Getting NUMA accesses right is difficult in a complex library. (Works fine in GHOST!)
- Strided accesses in row-major block vectors are slow. (Hidden by the interface.)

Implementation details

- Dedicated kernel routines for individual block sizes

Kernel routines

Performance pitfalls

- Common BLAS libraries are slow for the block vector GEMM operations here (e.g. $M \leftarrow X^T Y$).
- Getting NUMA accesses right is difficult in a complex library. (Works fine in GHOST!)
- Strided accesses in row-major block vectors are slow. (Hidden by the interface.)

Implementation details

- Dedicated kernel routines for individual block sizes
- Non-temporal stores where possible

Kernel routines

Performance pitfalls

- Common BLAS libraries are slow for the block vector GEMM operations here (e.g. $M \leftarrow X^T Y$).
- Getting NUMA accesses right is difficult in a complex library. (Works fine in GHOST!)
- Strided accesses in row-major block vectors are slow. (Hidden by the interface.)

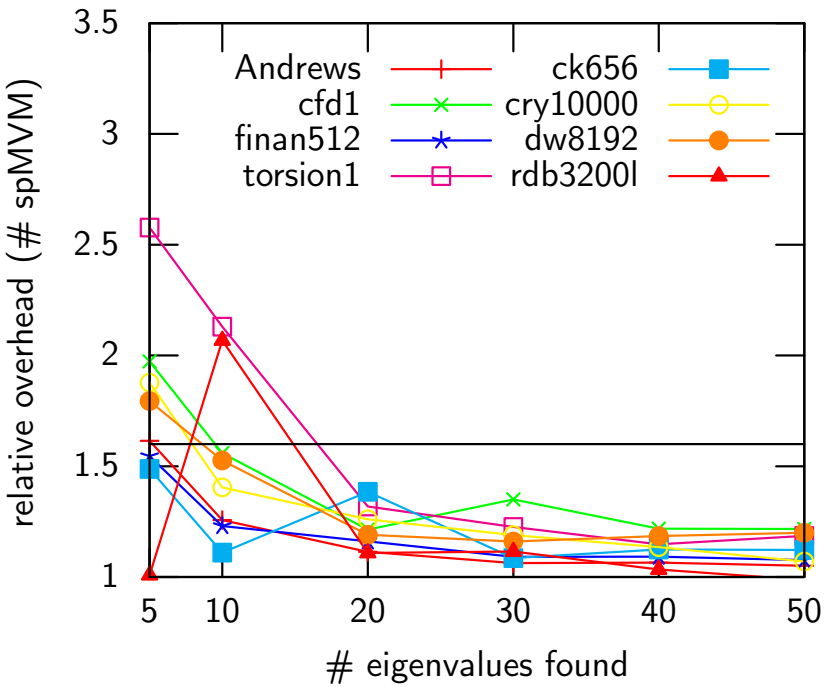
Implementation details

- Dedicated kernel routines for individual block sizes
- Non-temporal stores where possible
- SSE- / AVX-intrinsics

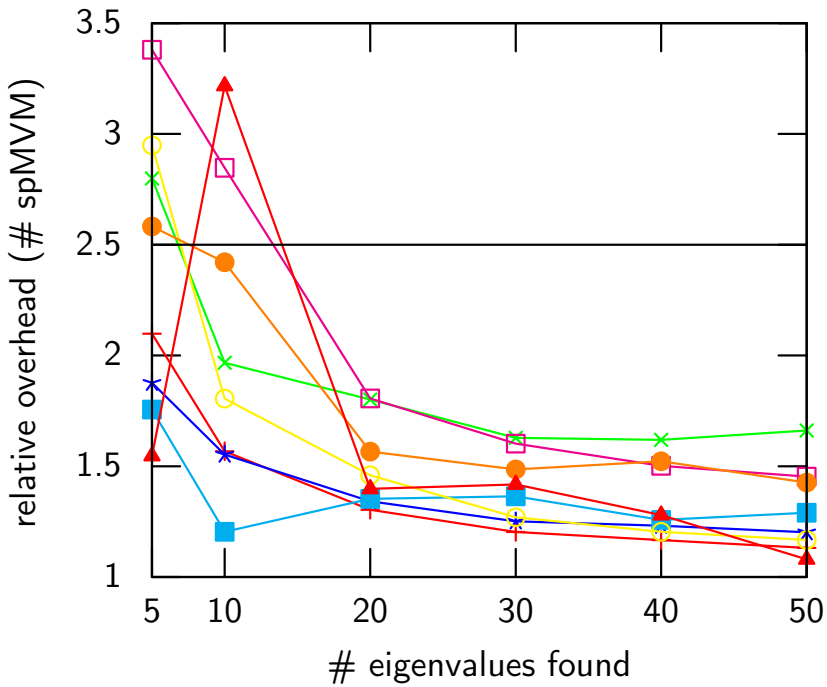
Outline

Numerical behavior

Block size 2

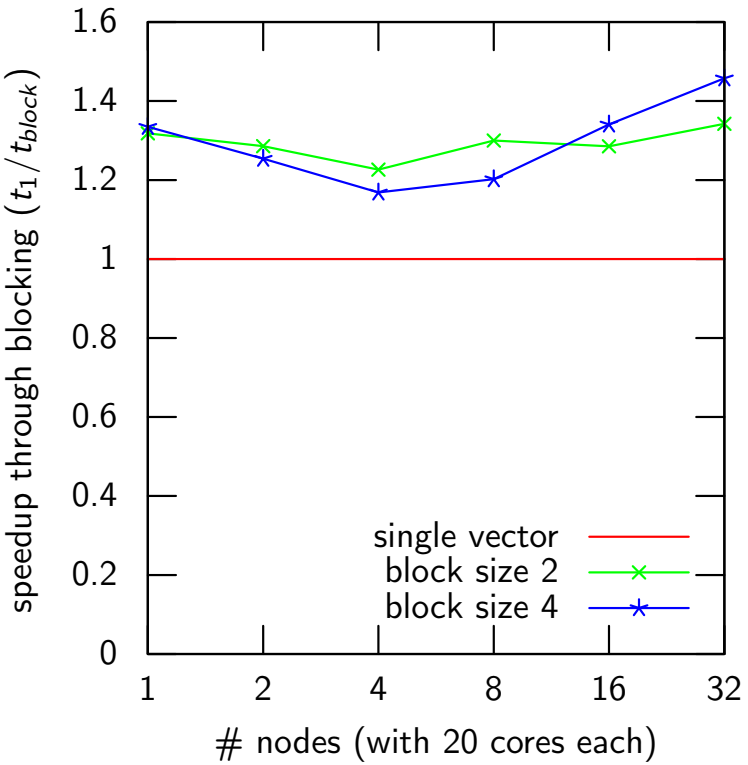


Block size 4

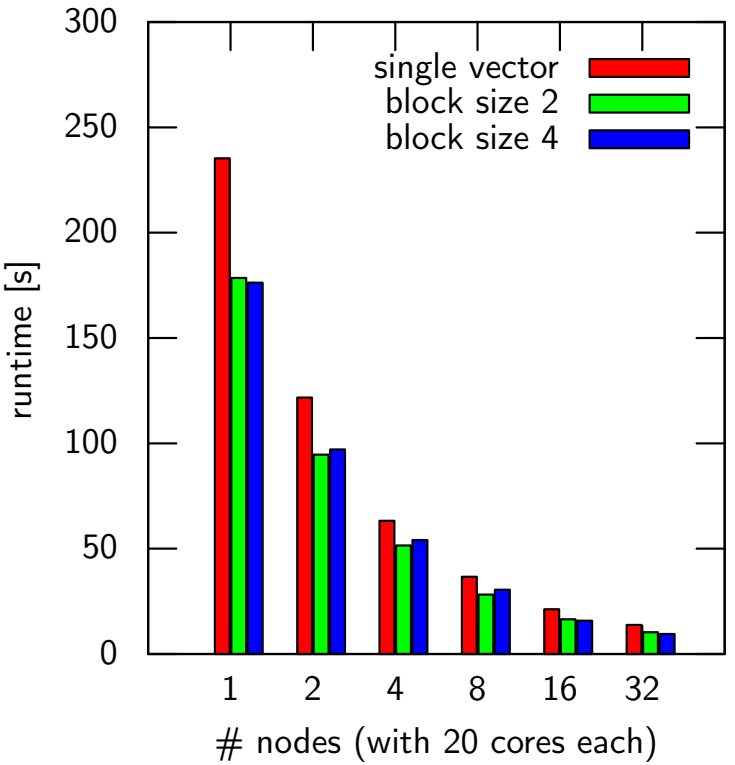


Performance of the complete algorithm (1)

Block speedup

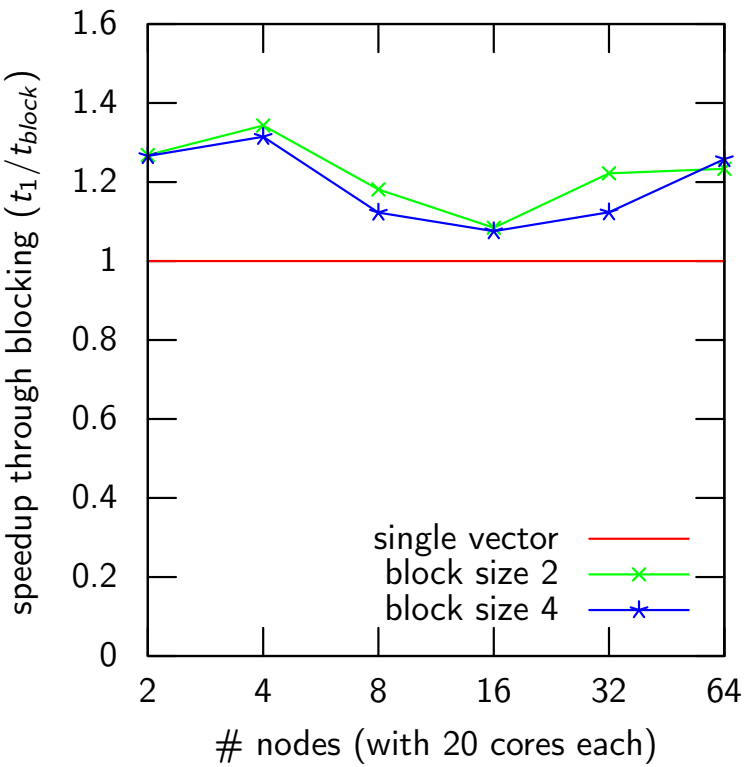


Strong scaling

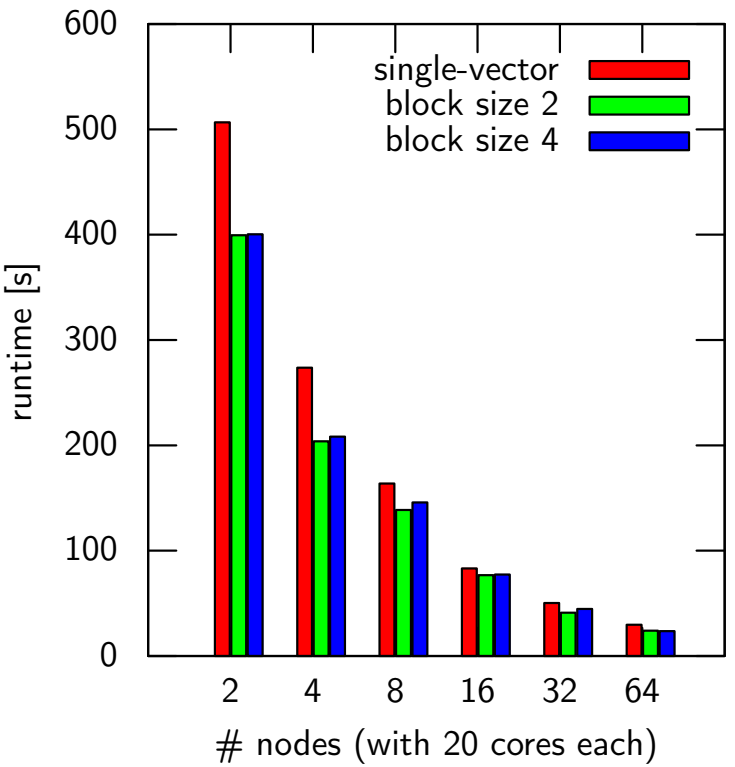


Performance of the complete algorithm (2)

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:
  - Specialized solver for symmetric problems
  - Parallel preconditioning of the linear problems

Conclusion (2)

Outlook

- Numerical improvements:
  - Specialized solver for symmetric problems
  - Parallel preconditioning of the linear problems
- Algorithmic improvements:
  - Hiding spMMVM communication behind other operations
  - More asynchronous communication (all-reductions)
  - Fast block orthogonalization (TSQR)

Conclusion (2)

Outlook

- Numerical improvements:
  - Specialized solver for symmetric problems
  - Parallel preconditioning of the linear problems
- Algorithmic improvements:
  - Hiding spMMVM communication behind other operations
  - More asynchronous communication (all-reductions)
  - Fast block orthogonalization (TSQR)
- Hybrid calculations with GPUs