Block Krylov and Jacobi-Davidson Methods on Heterogenous Systems

Jonas Thies[⋆] Dominik Ernst[†] Melven Röhrig-Zöllner[⋆]

[⋆] Institute of Simulation and Software Technology
German Aerospace Center (DLR)

[†] Erlangen Regional Computing Center (RRZE)
University of Erlangen-Nuremberg

SPPEXA

project ESSEX

DLR

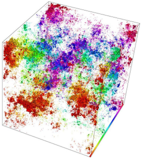
Sparse Eigenvalue Problems

Formulation: find some eigenpairs (λ_j, v_j) of a large and sparse matrix (pair) in a target region of the spectrum

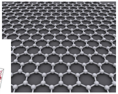
$$\mathbf{A}v_j = \lambda_j \mathbf{B}v_j$$

- **A** Hermitian or general, real or complex
- **B** may be identity matrix or Hermitian pos. def.

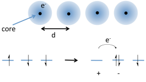
Applications



Anderson localization

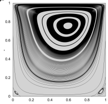


Graphene

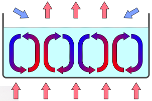


Hubbard model

Quantum and Fluid Mechanics



Driven cavity

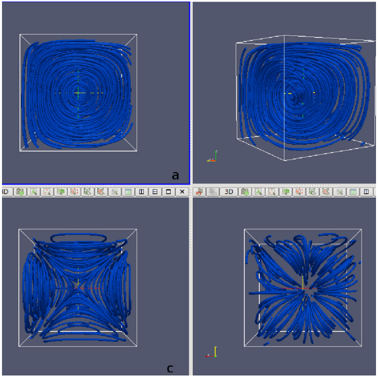


Rayleigh-Benard convection

DLR applications

Rayleigh-Bénard Convection

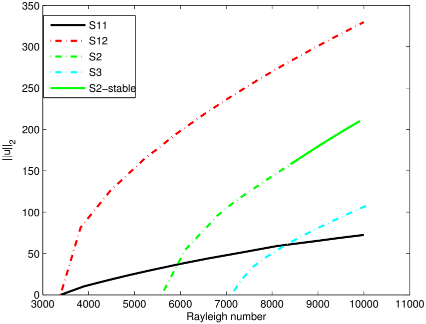
- Cube-shaped domain
- heated from below
- Rayleigh-Number
  $\mathrm{Ra} = \frac{\alpha g \Delta T d^3}{\nu \kappa}$



Flow patterns near the first three primary bifurcations
(a) x/y roll,
(b) diagonal roll,
(c) four rolls,
(d) toroidal roll

Rayleigh-Bénard Convection

- Cube-shaped domain
- heated from below
- Rayleigh-Number
  $\mathrm{Ra} = \frac{\alpha g \Delta T d^3}{\nu \kappa}$



Bifurcation diagram

In unstable steady states (dashed lines), the Jacobian has eigenvalues with positive real part.

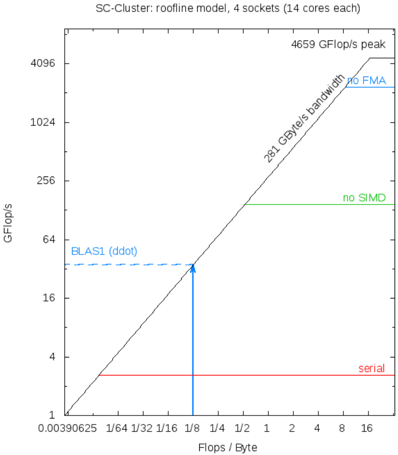
Performance of Sparse Matrix Algorithms

Typical operations are memory-bounded:

- 'spMVM' $y \leftarrow A \cdot x$,
- vector operations, $s \leftarrow x^T y$, $x \leftarrow \alpha x + \beta y$

... unless the data sets are small:

- CPU/KNL: OpenMP overhead $\approx 25\mu s$
- GPU: launch latency $\approx 35\mu s$



Target Hardware

- "Skylake": Intel Xeon Scalable, 4 × 14 cores @2.6GHz, **384 GB DDR4** RAM
- "KNL": Intel Xeon Phi, 64 cores @1.4GHz, 16 GB HBM (cache mode)
- "Volta": NVidia Tesla V100-SXM2 GPU, **16 GB HBM2** (+UVM)

Target Hardware

- “Skylake”: Intel Xeon Scalable, 4×14 cores @2.6GHz, **384 GB DDR4** RAM
- “KNL”: Intel Xeon Phi, 64 cores @1.4GHz, 16 GB HBM (cache mode)
- “Volta”: NVidia Tesla V100-SXM2 GPU, **16 GB HBM2** (+UVM)

benchmark	Skylake	KNL	Volta
load	360	338	812
store	200	167	883
triad	260	315	843

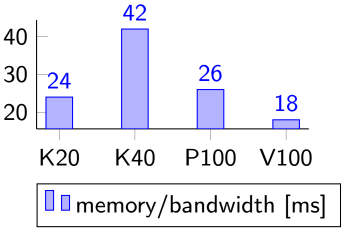
Measured streaming memory bandwidth [GB/s]

Target Hardware

- "Skylake": Intel Xeon Scalable, 4 × 14 cores @2.6GHz, **384 GB DDR4** RAM
- "KNL": Intel Xeon Phi, 64 cores @1.4GHz, 16 GB HBM (cache mode)
- "Volta": NVidia Tesla V100-SXM2 GPU, **16 GB HBM2** (+UVM)

benchmark	Skylake	KNL	Volta
load	360	338	812
store	200	167	883
triad	260	315	843

Measured streaming memory bandwidth [GB/s]



Target Hardware

- "Skylake": Intel Xeon Scalable, 4×14 cores @2.6GHz, **384 GB DDR4** RAM
- "KNL": Intel Xeon Phi, 64 cores @1.4GHz, 16 GB HBM (cache mode)
- "Volta": NVidia Tesla V100-SXM2 GPU, **16 GB HBM2** (+UVM)

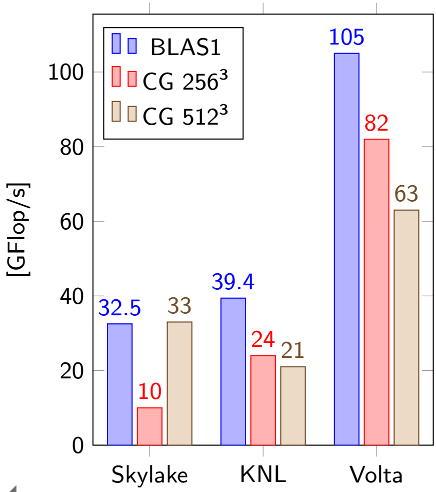
benchmark	Skylake	KNL	Volta
load	360	338	812
store	200	167	883
triad	260	315	843

Measured streaming memory bandwidth [GB/s]

n_b	1M	2M	4M	8M	16M	32M
1	12	23	37	58	78	83
2	31	35	53	68	81	88
4	34	53	66	83	88	95
8	51	70	85	87	99	100

"% roofline" of $X^T Y, X, Y \in \mathbb{R}^{N \times n_b}$ on Volta.

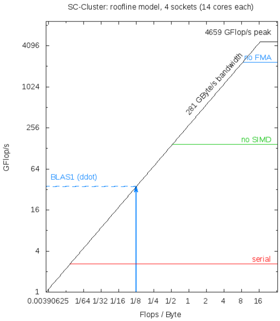
Example: Conjugate Gradients (CG)



- 1000 CG iterations
- 3D 7-point Laplacian

grid	N	memory
256^3	16.7M	2.2 GB
512^3	132M	18 GB

Increasing the Flop Intensity

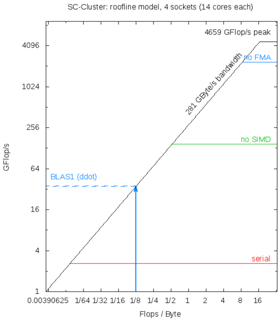


Block solvers (block size n_b)

- inner product $\Longrightarrow$ factor n_b^2 more flops
- vector updates remain BLAS1 ($X \leftarrow X + \alpha Y$)
- **Caveat:** may increase number of iterations
- **Example:** block GMRES for multiple RHS

Aim: push operations to the right

Increasing the Flop Intensity



Block solvers (block size n_b)

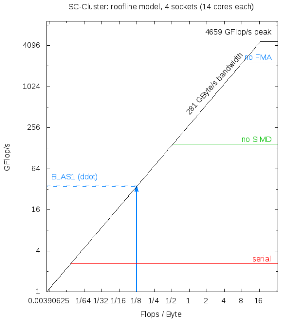
- inner product $\Longrightarrow$ factor n_b^2 more flops
- vector updates remain BLAS1 ($X \leftarrow X + \alpha Y$)
- **Caveat:** may increase number of iterations
- **Example:** block GMRES for multiple RHS

Kernel fusion

- example: compute $Y \leftarrow AX$ and simultaneously $C = X^T Y$ 'for free'
- requires specialized kernels
- no deterioration of numerics

Aim: push operations to the right

Increasing the Flop Intensity



Aim: push operations to the right

Block solvers (block size n_b)

- inner product $\Longrightarrow$ factor n_b^2 more flops
- vector updates remain BLAS1 ($X \leftarrow X + \alpha Y$)
- **Caveat:** may increase number of iterations
- **Example:** block GMRES for multiple RHS

Kernel fusion

- example: compute $Y \leftarrow AX$ and simultaneously $C = X^T Y$ 'for free'
- requires specialized kernels
- no deterioration of numerics

Mixed Precision (work in progress)

- store (block) vectors in single precision
- compute in double to maintain numerical stability
- also allows larger problems on GPU

Example: Block Orthogonalization

Problem definition

- Given orthogonal vectors $(w_1, \ldots, w_k) = W$
- For $X \in \mathbb{R}^{n \times n_b}$ find orthogonal $Y \in \mathbb{R}^{n \times \tilde{n}_b}$ with

$$YR_1 = X - WR_2, \qquad \text{and} \qquad W^T Y = 0$$

Two phase algorithms

Phase 1 Project: $\bar{X} \leftarrow (I - WW^T)X$

Phase 2 Orthogonalize: $Y \leftarrow f(\bar{X})$

- suitable f:
  - SVQB (Stathopoulos and Wu, SISC 2002)
  - TSQR (Demmel et al., SISC 2012)
- Each phase messes with the accuracy of the other. → iterate

Block Orthogonalization with Kernel Fusion

Rearrange and fuse operations to reduce memory traffic:

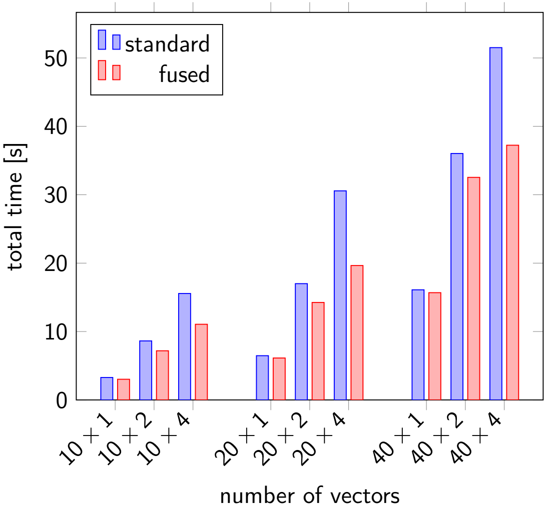
Phase 2 $\bar{X} \leftarrow X\bar{M}, \qquad N \leftarrow W^T\bar{X}$

Phase 1 $\bar{X} \leftarrow X - WN, \quad M \leftarrow \bar{X}^T\bar{X}$

Phase 3 $\bar{X} \leftarrow X\bar{M}, \qquad M \leftarrow \bar{X}^T\bar{X}$

$\Rightarrow$ use SVQB or Cholesky-QR

Block Orthog: runtime to convergence



Software I: our Kernel Library

General, **H**ybrid-parallel and
Optimized **S**parse **T**oolkit

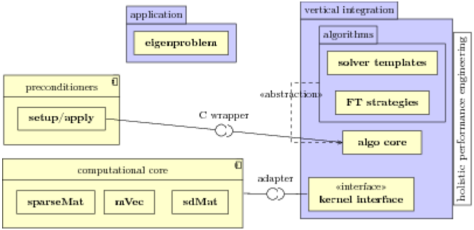
- provides memory-bounded kernels for sparse solvers
- **data structures:**
  - row- or col-major block vectors
  - SELL-$C-\sigma$ for sparse matrices
- written (mostly) in C
- **'MPI+X'** with X OpenMP, CUDA and SIMD intrinsics
- runs on Peta-scale systems (Piz Daint, Oakforest-PACS)
- can use heterogenous systems (e.g. including CPUs, MIC and GPUs)

`https://bitbucket.org/essex/ghost`

Software II: Algorithms and Integration Framework

PHIST

Pipelined, **H**ybrid-parallel
Iterative **S**olver **T**oolkit



- Interfaces: C, C++, Fortran, Python
- testing and benchmarking tools
- includes performance models
- various linear and eigensolvers

Select 'backend' at compile time:
GHOST, builtin (Fortran), Trilinos, PETSc

https://bitbucket.org/essex/phist

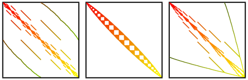
Software III: Eigensolver Benchmark Matrices

ScaMaC

Scalable **Ma**trix **C**ollection

- scalable matrices, scalable generators
- "real world" matrices mainly from classical & quantum physics
  wave & advection-diffusion eqs., correlated (fermion, boson, spin) systems, graphene&topological insulators, quantum optics, (c)QED, optomechanics, …
- real & complex, symmetric & non-symmetric, small → huge, easy & hard matrices
- stand-alone or as part of PHIST

exemplary sparsity patterns



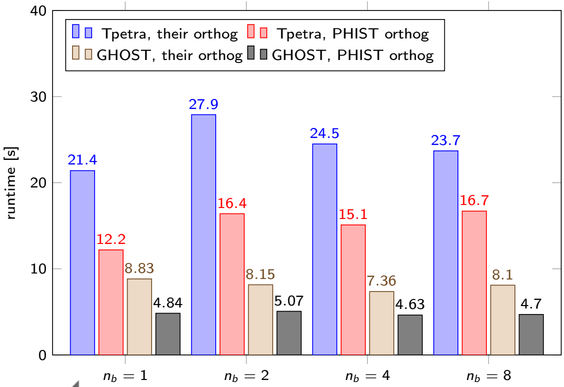
```
=== example:      FermionChain12 ===
=== parameter:    n_fermions      ===
```

n_fermions	matrix dimension
5	252
6	924
7	3 432
⋮	
18	9 075 135 300
19	35 345 263 800
20	137 846 528 820
⋮	

https://bitbucket.org/essex/MatrixCollection

Example: Anasazi Block Krylov-Schur on Skylake CPU

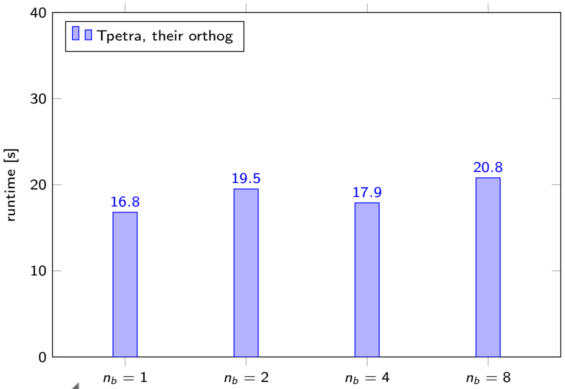
Matrix: non-symmetric 7-point stencil, $N = 128^3$
(var. coeff. reaction/convection/diffusion)



- Anasazi's kernel interface is mostly a subset of PHIST's
  $\Longrightarrow$ extends PHIST by e.g. BKS and LOBPCG
- not optimized for block vectors in row-major storage

Example: Anasazi Block Krylov-Schur on Volta GPU

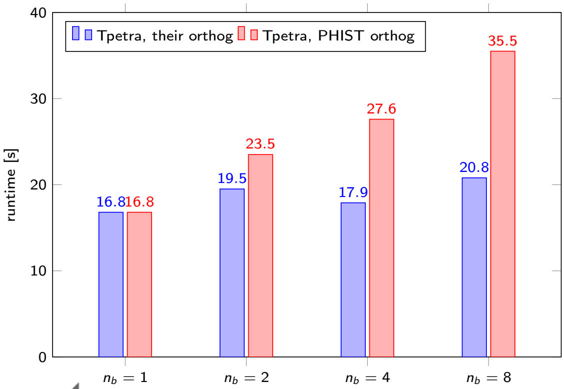
Matrix: non-symmetric 7-point stencil, $N = 128^3$
(var. coeff. reaction/convection/diffusion)



- Anasazi's kernel interface is mostly a subset of PHIST's
  $\Longrightarrow$ extends PHIST by e.g. BKS and LOBPCG
- not optimized for block vectors in row-major storage

Example: Anasazi Block Krylov-Schur on Volta GPU

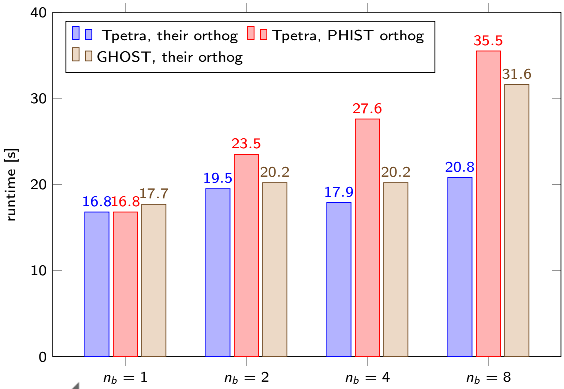
Matrix: non-symmetric 7-point stencil, $N = 128^3$
(var. coeff. reaction/convection/diffusion)



- Anasazi's kernel interface is mostly a subset of PHIST's
  $\Longrightarrow$ extends PHIST by e.g. BKS and LOBPCG
- not optimized for block vectors in row-major storage

Example: Anasazi Block Krylov-Schur on Volta GPU

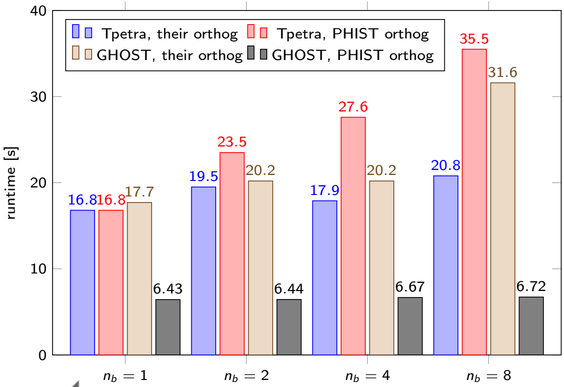
Matrix: non-symmetric 7-point stencil, $N = 128^3$
(var. coeff. reaction/convection/diffusion)



- Anasazi's kernel interface is mostly a subset of PHIST's
  $\Longrightarrow$ extends PHIST by e.g. BKS and LOBPCG
- not optimized for block vectors in row-major storage

Example: Anasazi Block Krylov-Schur on Volta GPU

Matrix: non-symmetric 7-point stencil, $N = 128^3$
(var. coeff. reaction/convection/diffusion)



- Anasazi's kernel interface is mostly a subset of PHIST's
  $\Longrightarrow$ extends PHIST by e.g. BKS and LOBPCG
- not optimized for block vectors in row-major storage

Block Jacobi-Davidson QR

- Use *inexact Newton* rather than Krylov sequence
- *JDQR:* subspace acceleration, locking and restart (Fokkema'99)

Block Jacobi-Davidson correction equation

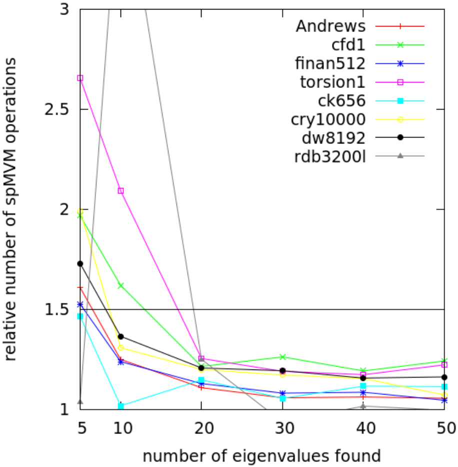
- n_b current approximations: $A\tilde{v}_i - \tilde{\lambda}_i \tilde{v}_i = r_i$, $i = 1, \dots, n_b$
- previously converged Schur vectors $\left(q_1, \dots, q_k\right) = Q$
- solve approximately (with $\tilde{Q} = \begin{pmatrix} Q & \tilde{v}_1 & \dots & \tilde{v}_{n_b} \end{pmatrix}$):

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

- use some steps of a block(ed) iterative solver
- orthogonalize new directions $x_1, \dots, x_{n_b}$ (outer subspace iteration)

BJDQR: 'Numerical Overhead'

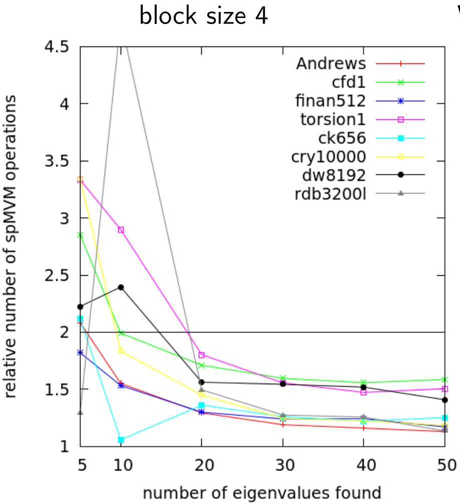
block size 2



With larger block size...

- number of (outer) iteratios decreases
- total number of operations increases
- tested here for various matrices

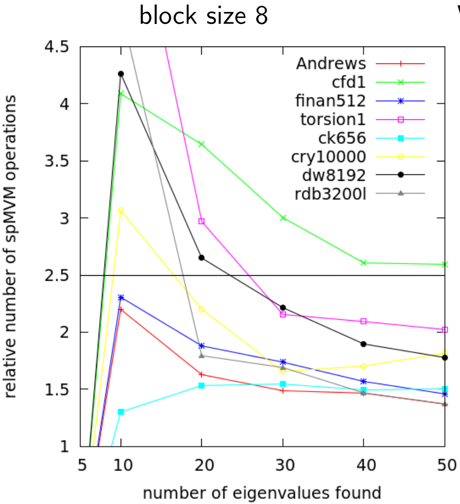
BJDQR: 'Numerical Overhead'



With larger block size...

- number of (outer) iteratios decreases
- total number of operations increases
- tested here for various matrices

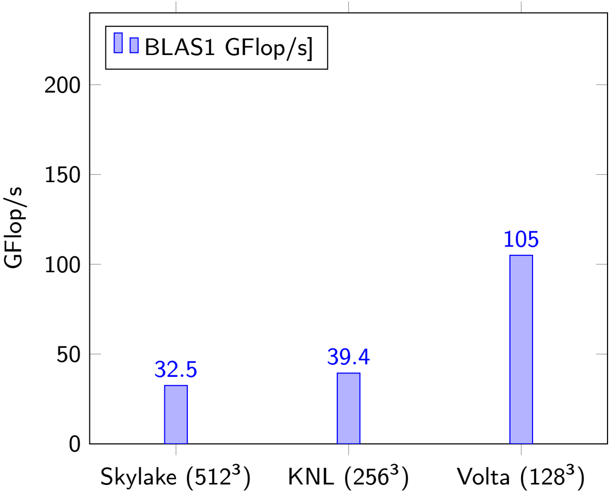
BJDQR: 'Numerical Overhead'



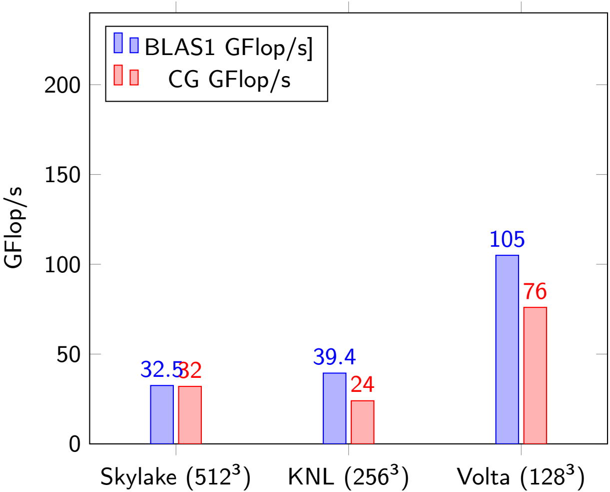
With larger block size...

- number of (outer) iteratios decreases
- total number of operations increases
- tested here for various matrices

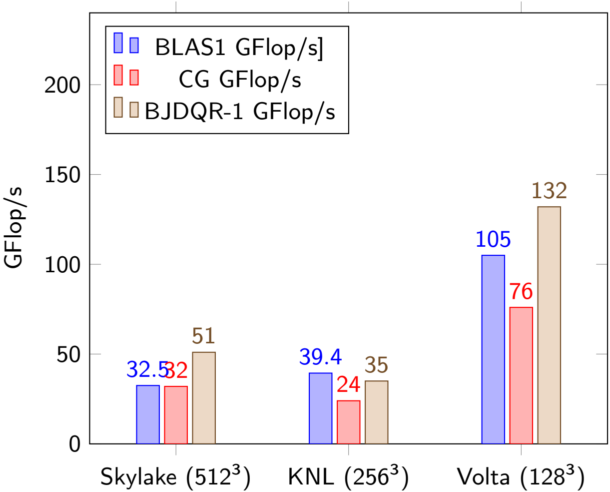
BJDQR on Different Hardware (here: Laplace problem)



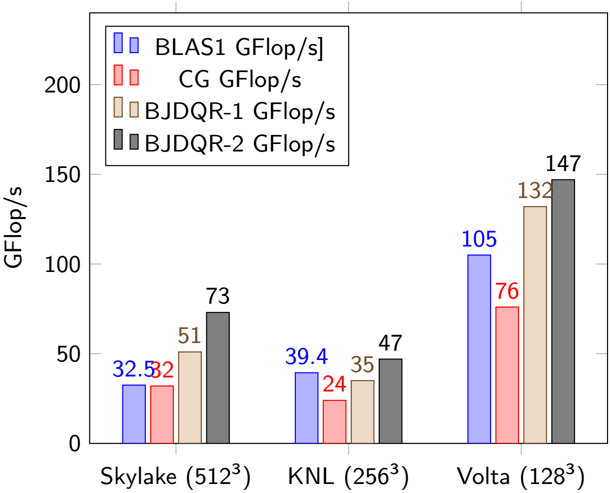
BJDQR on Different Hardware (here: Laplace problem)



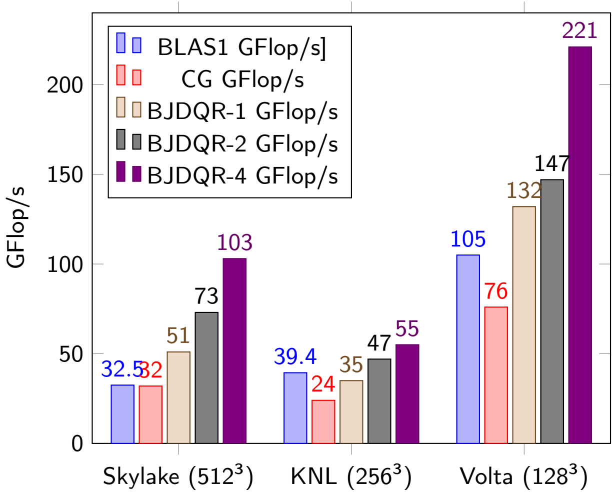
BJDQR on Different Hardware (here: Laplace problem)



BJDQR on Different Hardware (here: Laplace problem)



BJDQR on Different Hardware (here: Laplace problem)



Preconditioning with ML

- non-symm. PDE as before
- use AMG preconditioner ML from Trilinos ("non-symmetric smoothed aggregation")

problem size	preconditioner	iterations	spMVMs	t_{tot}	t_{gmres}
128^3	GMRES	447	9 875	52.6s	25.0s
	GMRES+ML	26	612	21.2s	11.4s
256^3	GMRES	781	17 223	1 300s	571s
	GMRES+ML	40	922	346s	183s
512^3	GMRES	>1k	>22k	>1h	>1h
	GMRES+ML	32	746	624s	320s

Summary

- Three libraries for high-performance eigenvalue solvers
- intended to be useful to the algorithms community!

Our experience with GPUs so far

- (fast) memory is scarce
- but good performance reuires very long vectors
- block algorithms require (near) roofline performance to 'pay off'
- and more memory than their scalar counterparts

Next steps:

- investigate strong scaling (block methods expected to perform well)
- IBM/Volta may be an interesting architecture to explore
- hybrid-parallel preconditioners

https://bitbucket.org/essex/[ghost|phist|MatrixCollection]