High Performance Sparse Linear Algebra Libraries: GHOST and PHIST

Jonas Thies

German Aerospace Center (DLR)
Simulation and Software Technology
High Performance Computing
Jonas.Thies@DLR.de

project ESSEX

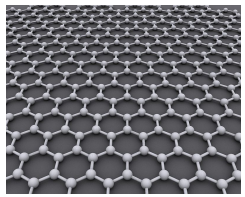
Sparse Eigenvalue Problems

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

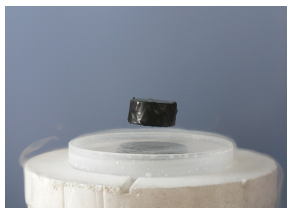
$$\mathbf{A}v_j = \lambda_j \mathbf{B}v_j$$

- **A** Hermitian or general, real or complex
- **B** may be identity matrix (or not)
- 'some': 10–100 or so

Applications:

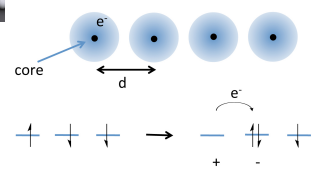


Graphene

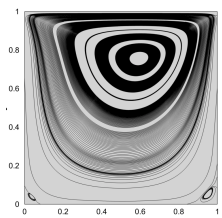


Superconductors

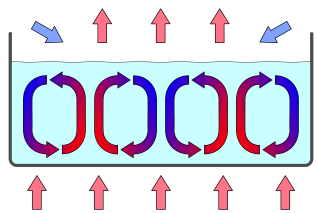
Quantum and Fluid Mechanics



Hubbard model



Driven cavity



Rayleigh-Benard convection

DLR applications

pictures taken from `wikipedia.org`

Application: Linear Stability Analysis for Dynamical Systems

- steady state is **linearly stable** if all eigenvalues of J are negative
- **bifurcation points** are locations in parameter space where the spectral properties of J change
- **branch switching:** force system in the direction of an unstable eigenvector at a bifurcation point

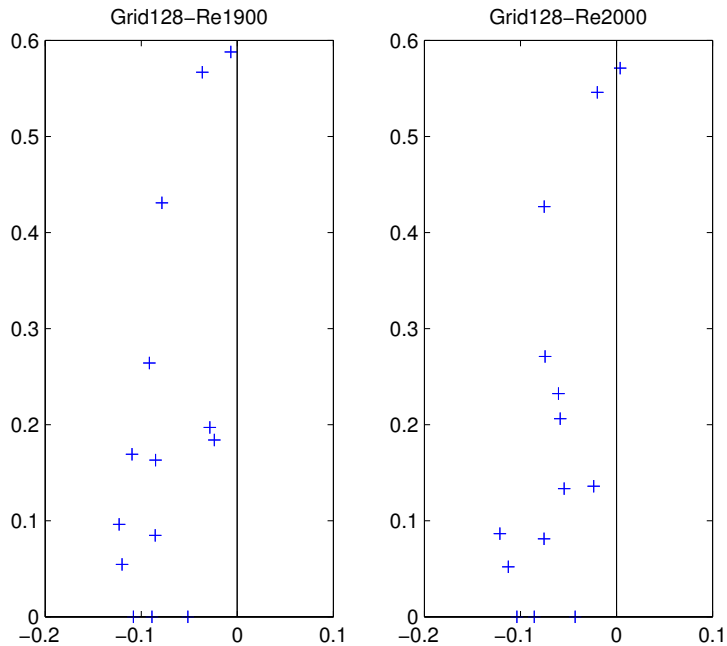
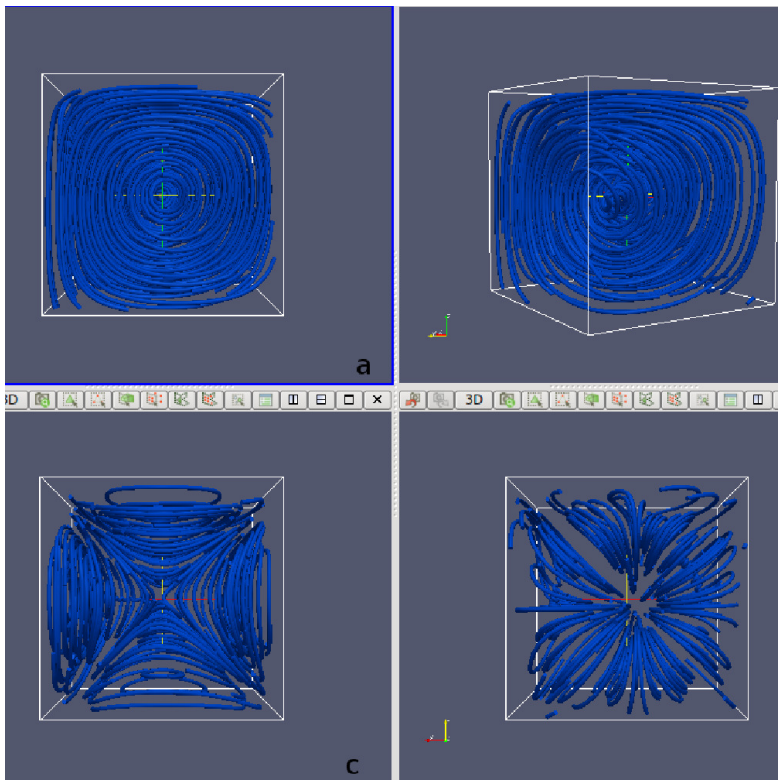


Figure: Hopf bifurcation in 3D lid-driven cavity flow

Example: Bifurcation Analysis of 3D Rayleigh-Benárd 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



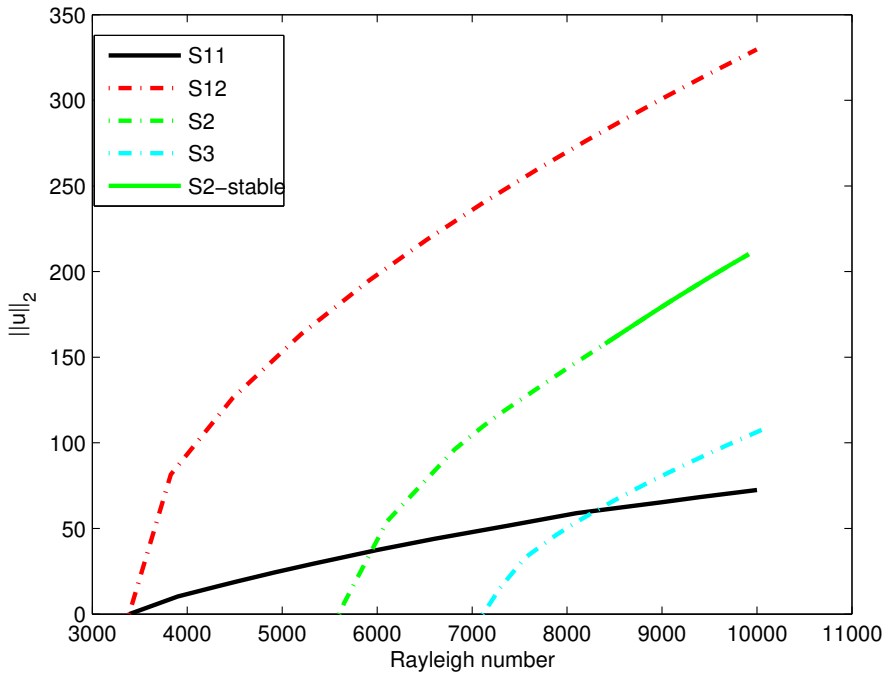
Example: Bifurcation Analysis of 3D Rayleigh-Benárd Convection

- 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.



Block Jacobi-Davidson QR

- Aim: partial QR decomposition, $\mathrm{A}Q = QR$, $R \in \mathbb{C}^{k \times k}$ upper triangular,
- $\frac{1}{2}Q^T Q - \frac{1}{2}I = 0, Q \in \mathbb{R}^{N \times k}$.

Newton's method, let $Q = \tilde{Q} + \Delta Q$

- $A\Delta Q - \Delta Q\tilde{R} = A\tilde{Q} - \tilde{Q}\tilde{R}$
- $\tilde{Q}^T \Delta Q = 0$

Block Jacobi-Davidson QR (2)

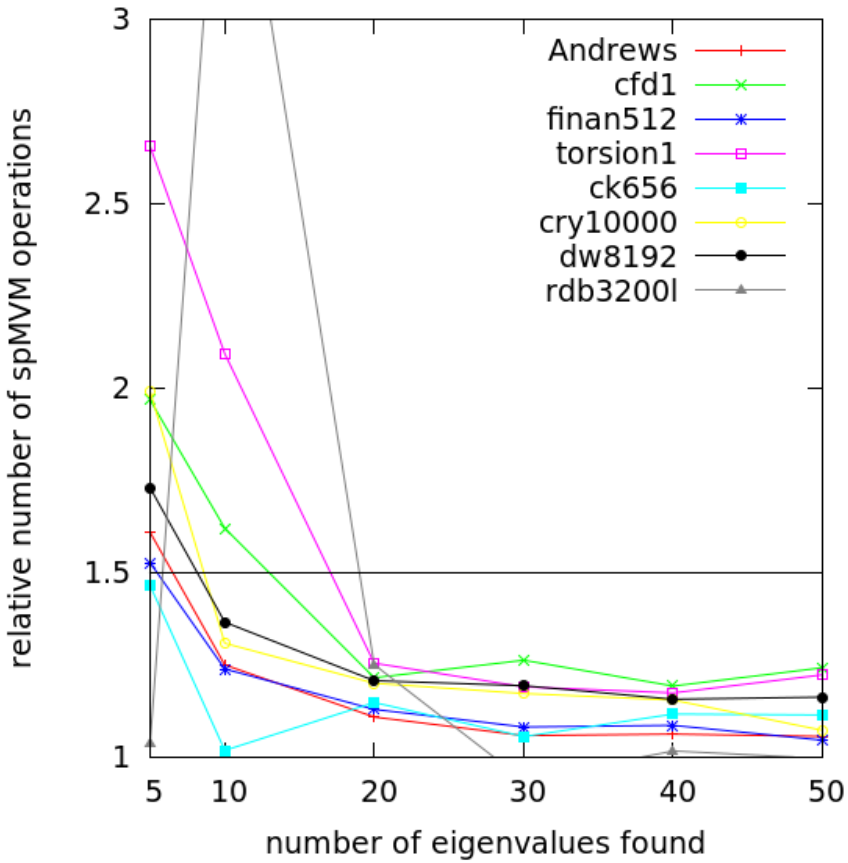
This leads to a set of *correction equations*

$$(I - \tilde{Q}\tilde{Q}^T)A(I - \tilde{Q}\tilde{Q}^T)\Delta Q - \Delta Q\tilde{R} = A\tilde{Q} - \tilde{Q}\tilde{R}$$

- Subspace acceleration: add corrections to expanding search space V
- Ritz-Galerkin: $M = V^T \mathrm{A} V$, $M = S^H R S$
- Lock converged eigenpairs $\Rightarrow$ growing projection space $\tilde{Q}$
- Solve correction eq. using (deflated) GMRES or MINRES Krylov solver
- Restart with $m_{min} > n_{eigs}$ vectors when basis becomes too large

BJDQR: 'Numerical Overhead'

block size 2

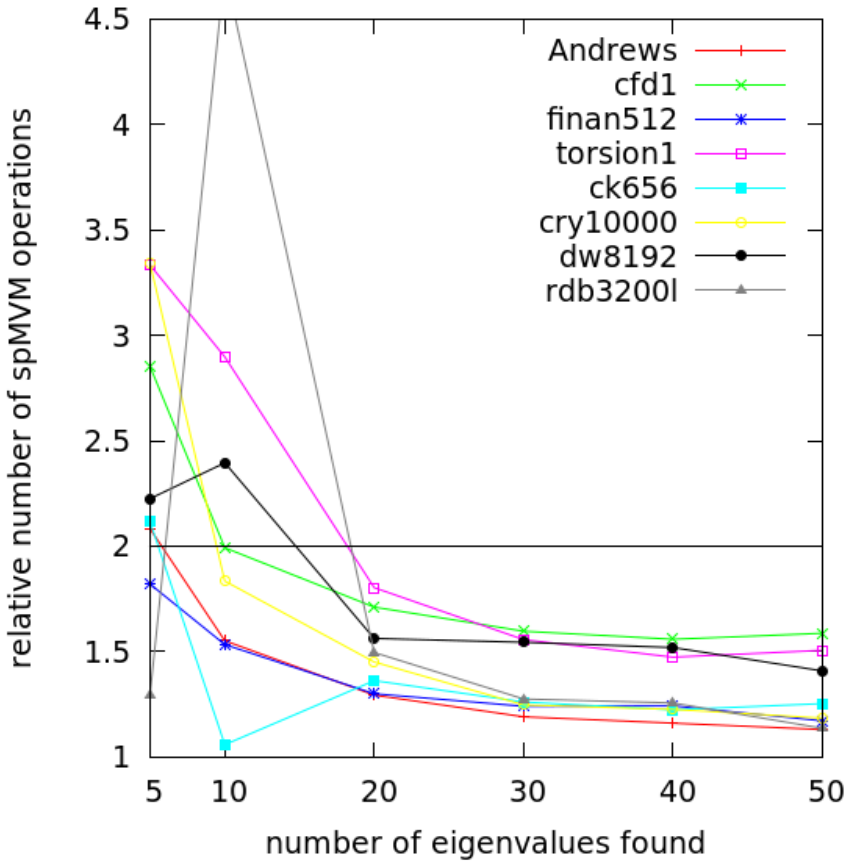


With larger block size...

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

BJDQR: 'Numerical Overhead'

block size 4

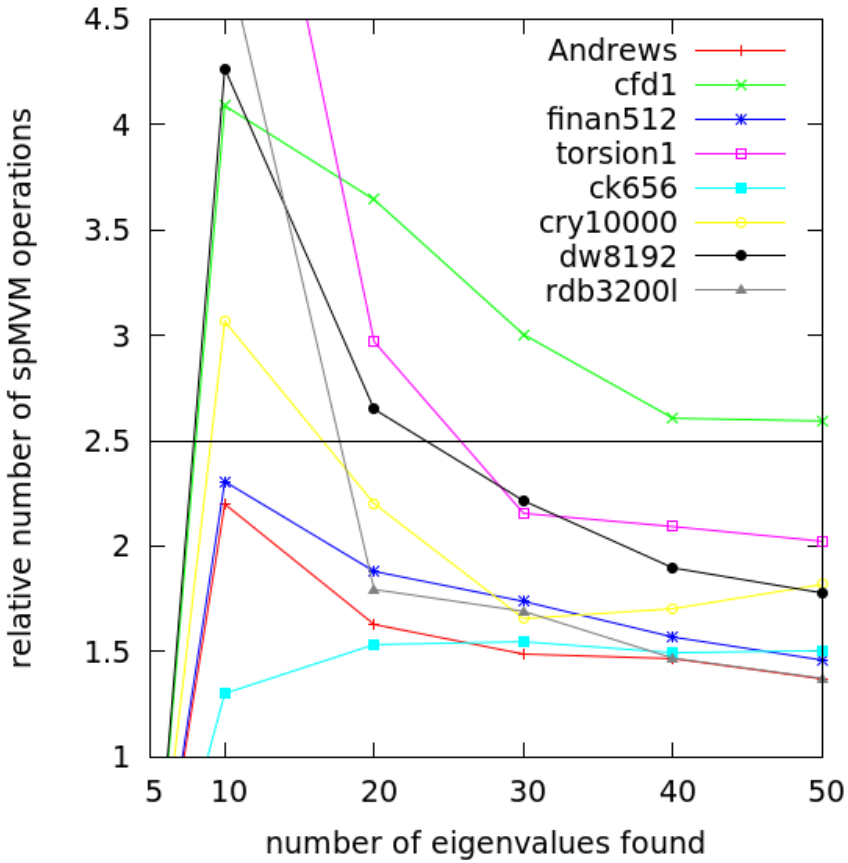


With larger block size...

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

BJDQR: 'Numerical Overhead'

block size 8



With larger block size...

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

BJDQR enables preconditioning for EVP

Effect of using an AMG preconditioner with BJDQR

Matrix: non-symmetric 3D PDE problem
Preconditioner: Trilinos ML 'NSSA'
(non-symmetric smoothed aggregation)

problem size	preconditioner	iterations	spMVMs	t_{tot}	t_{gmres}
128^3	GMRES	471	10 403	38.5	24.7
	GMRES+ML	31	720	26.3	13.2
256^3	GMRES	815	17 971	736	496
	GMRES+ML	29	668	227	116

Software I: our Kernel Library

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

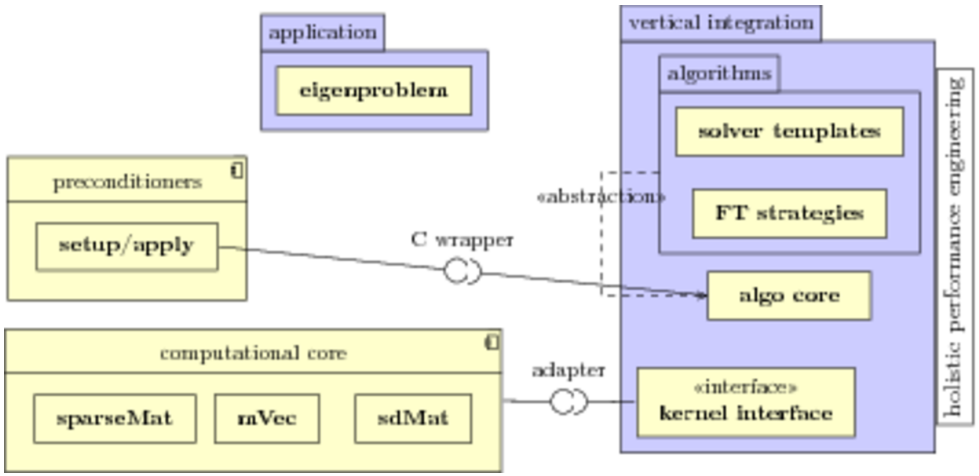
- provides memory-bounded kernels for sparse solvers
- **data structures:**
  - row- or col-major block vectors
  - SELL-C-σ for sparse matrices
- written (mostly) in C
- **'MPI+X'** with X OpenMP, CUDA and SIMD intrinsics
- runs on Peta-scale systems (Piz Daint, Oakforest-PACS)
- can use heterogeneous 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`

DLR

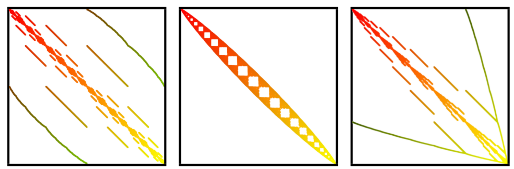
Software III: Eigensolver Benchmark Matrices

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 ===Âă
=== 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

PHIST in the Software Landscape

- Use different kernel libraries depending on application and hardware
- Available interfaces: C, C++, Fortran'03 and Python
- Can use Trilinos solvers with GHOST or other kernels (e.g. block CG/GMRES, Krylov-Schur, LOBPCG)
- Installation of PHIST and GHOST via the SPACK package manager (`https://github.com/spack/spack`)

- will join the extreme-Scale Development Kit (`https://xSDK.info`)

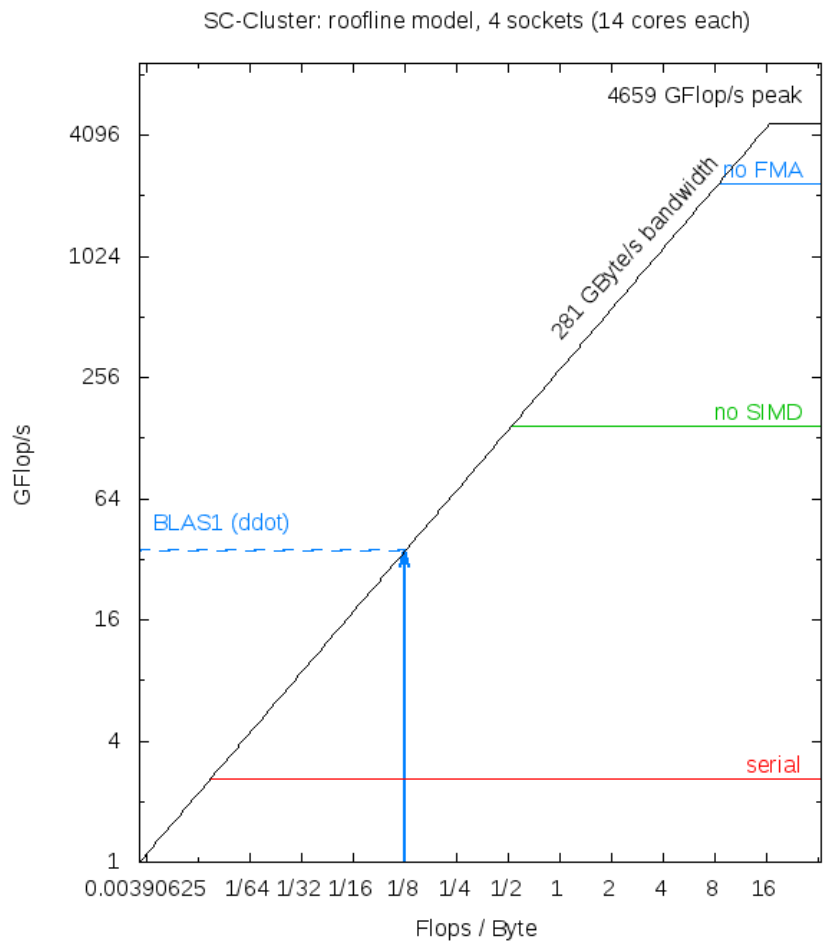
Performance of Sparse Matrix Algorithms

Typical operations are memory-bounded:

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

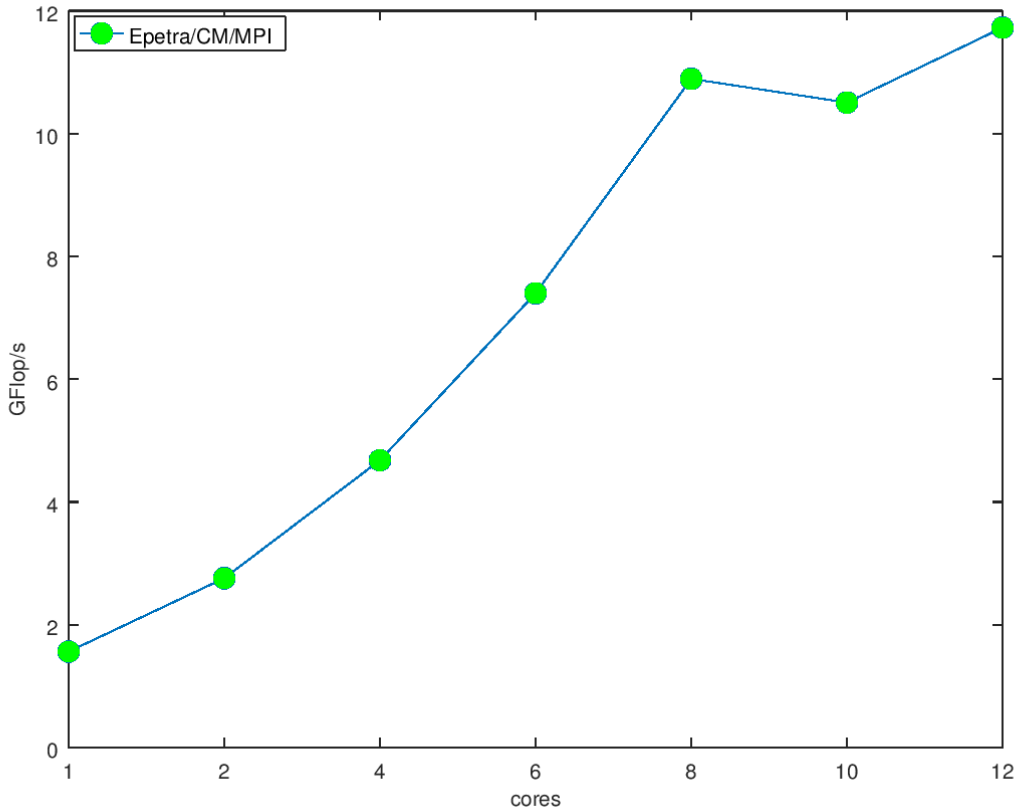
... unless the data sets are small:

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



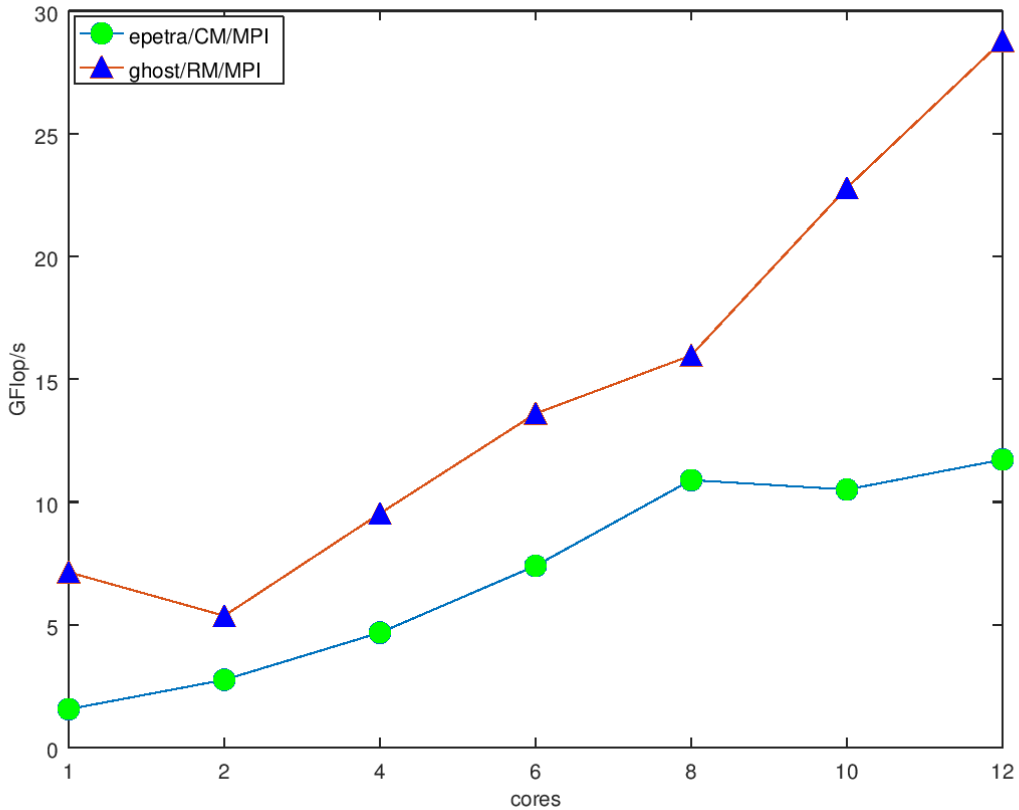
Why Performance Engineering?

simple(?) operation: $C = V^T V, V \in \mathbb{R}^{1M \times 4}$ on an Intel Haswell CPU



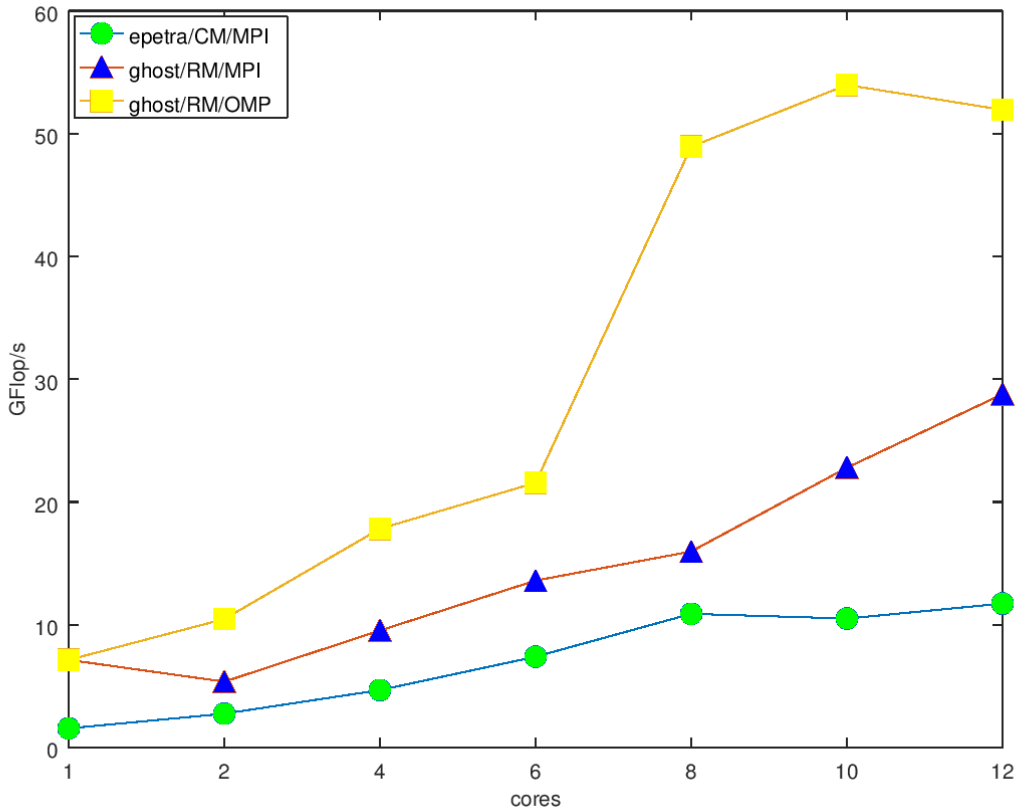
Why Performance Engineering?

simple(?) operation: $C = V^T V, V \in \mathbb{R}^{1M \times 4}$ on an Intel Haswell CPU



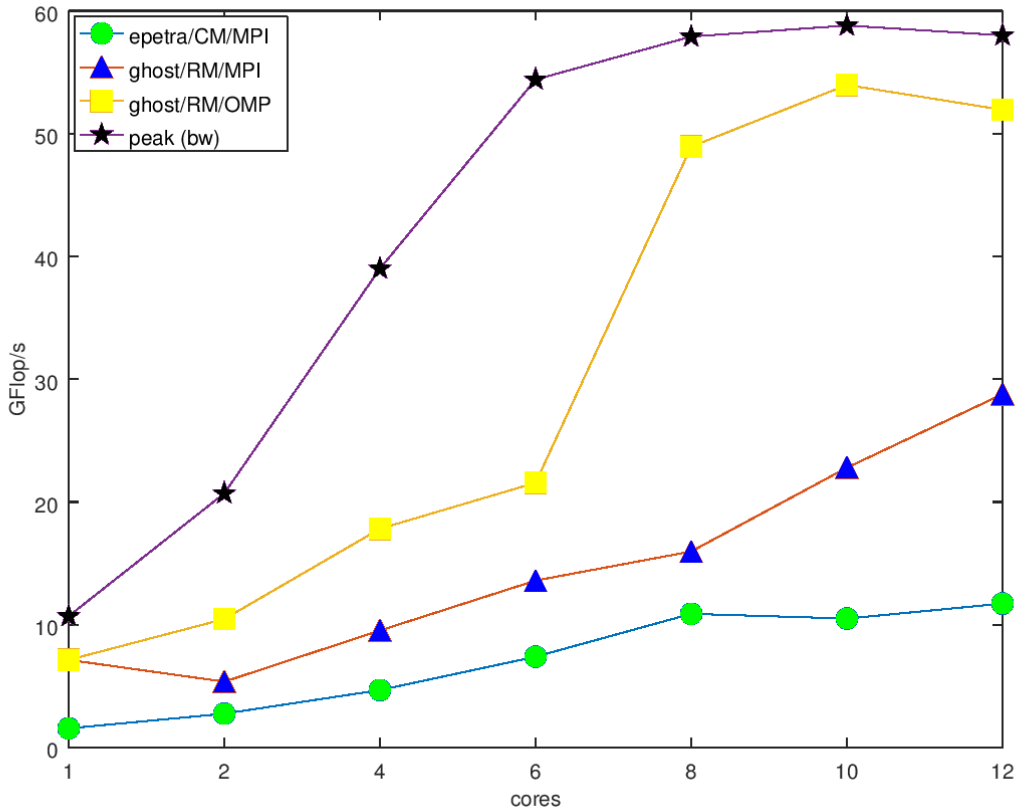
Why Performance Engineering?

simple(?) operation: $C = V^T V, V \in \mathbb{R}^{1M \times 4}$ on an Intel Haswell CPU



Why Performance Engineering?

simple(?) operation: $C = V^T V, V \in \mathbb{R}^{1M \times 4}$ on an Intel Haswell CPU



Performance on Different Hardware

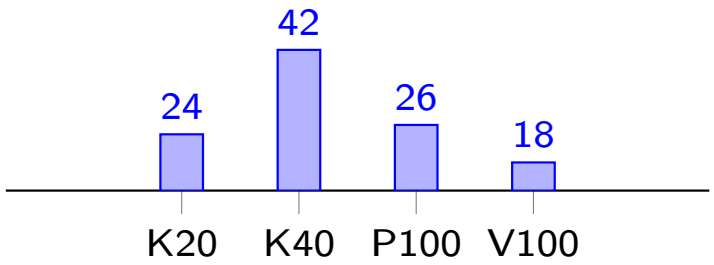
Test Hardware

- "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]

GPUs are increasingly hungry

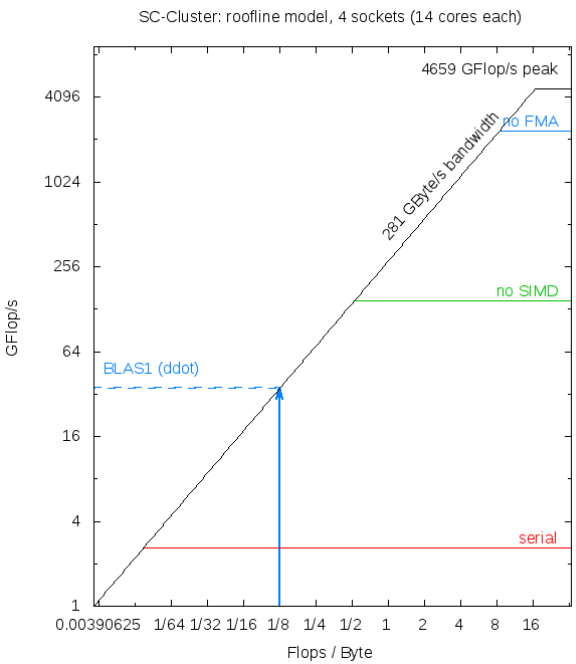


Ratio of GPU memory and memory bandwidth [ms] over time.

bs	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 $X^T Y, X, Y \in \mathbb{R}^{N \times bs}$ on Volta using GHOST

Increasing the Flop Intensity

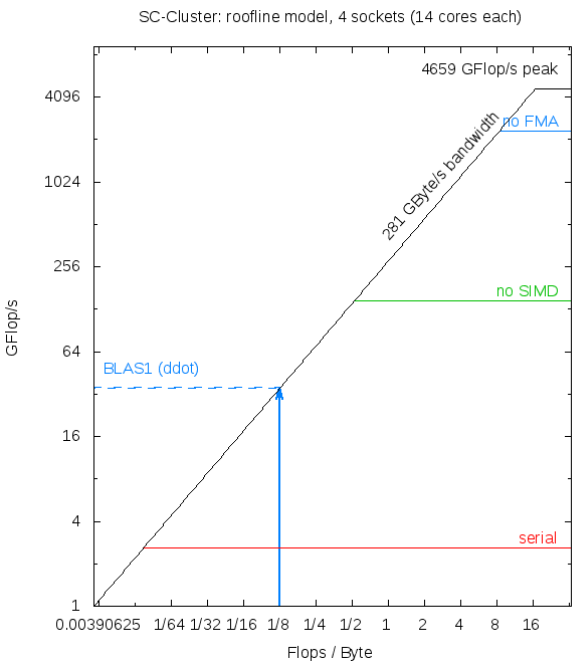


Block solvers (block size n_b)

- 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)

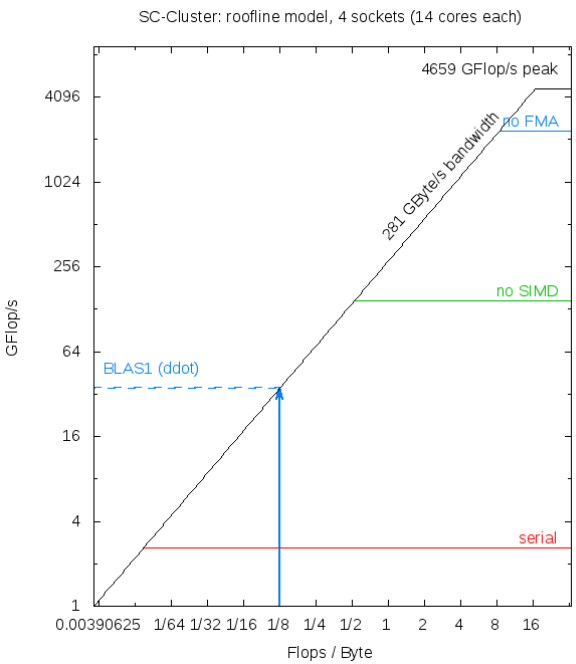
- inner product $\Longrightarrow$ factor n_b^2 more flops
- vector updates remain BLAS1 ($X \leftarrow X + \alpha Y$)
- **Caveat:** may increase number of iterations
- **Example:** block GMRES for multiple RHS

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 $\left(w_1, \ldots, w_k\right) = W$
- 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. $\rightarrow$ iterate

Block Orthogonalization with Kernel Fusion

Rearrange and fuse operations to reduce memory traffic:

$$\begin{aligned}
&\text{Phase 2} && \bar{X} \leftarrow X\bar{M}, && N \leftarrow W^T\bar{X} \\
&\text{Phase 1} && \bar{X} \leftarrow X - WN, && M \leftarrow \bar{X}^T\bar{X} \\
&\text{Phase 3} && \bar{X} \leftarrow X\bar{M}, && M \leftarrow \bar{X}^T\bar{X}
\end{aligned}$$

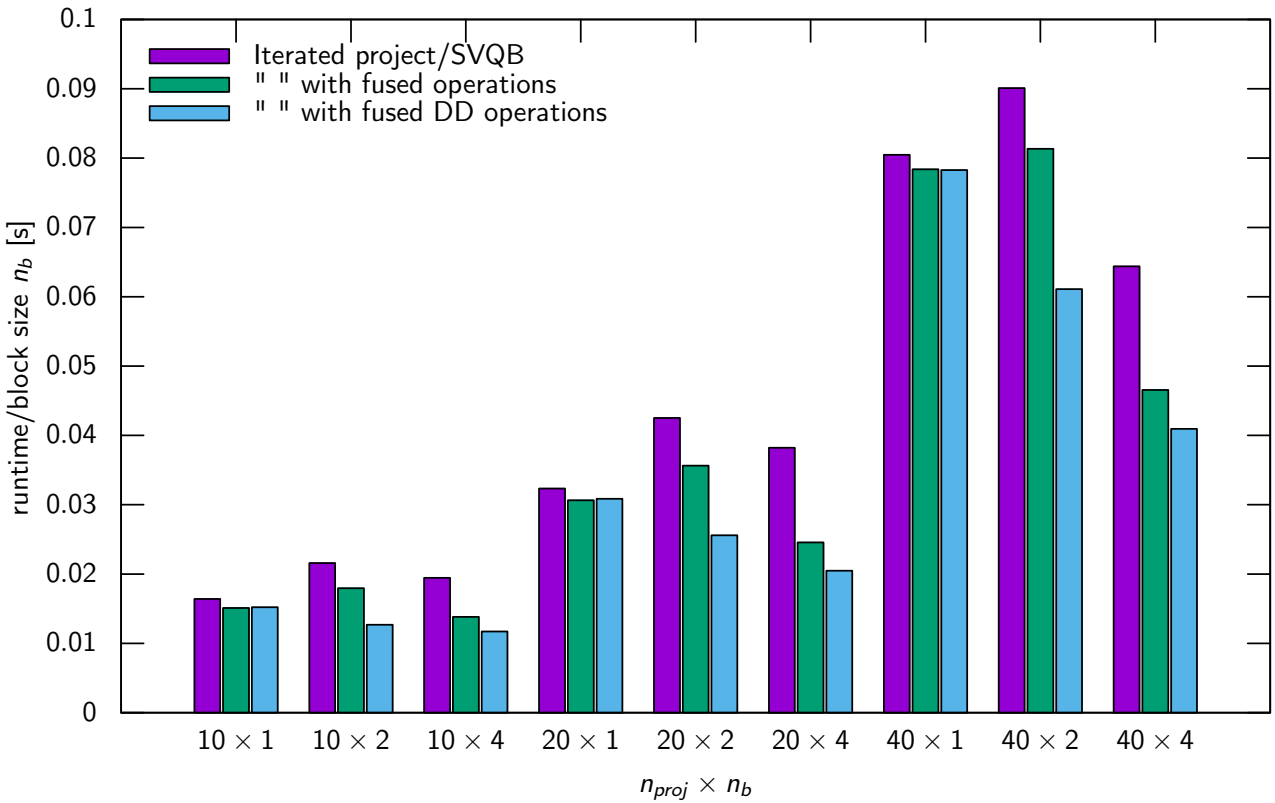
$\Rightarrow$ use SVQB or Cholesky-QR

Increased precision

Idea Calculate value and error of each arithmetic operation

- Store intermediate results as double-double (DD) numbers
- Based on arithmetic building blocks (2Sum, 2Mult)
  Muller et al.: Handbook of Floating-Point Arithmetic, Springer 2010
- Exploit FMA operations (AVX2)

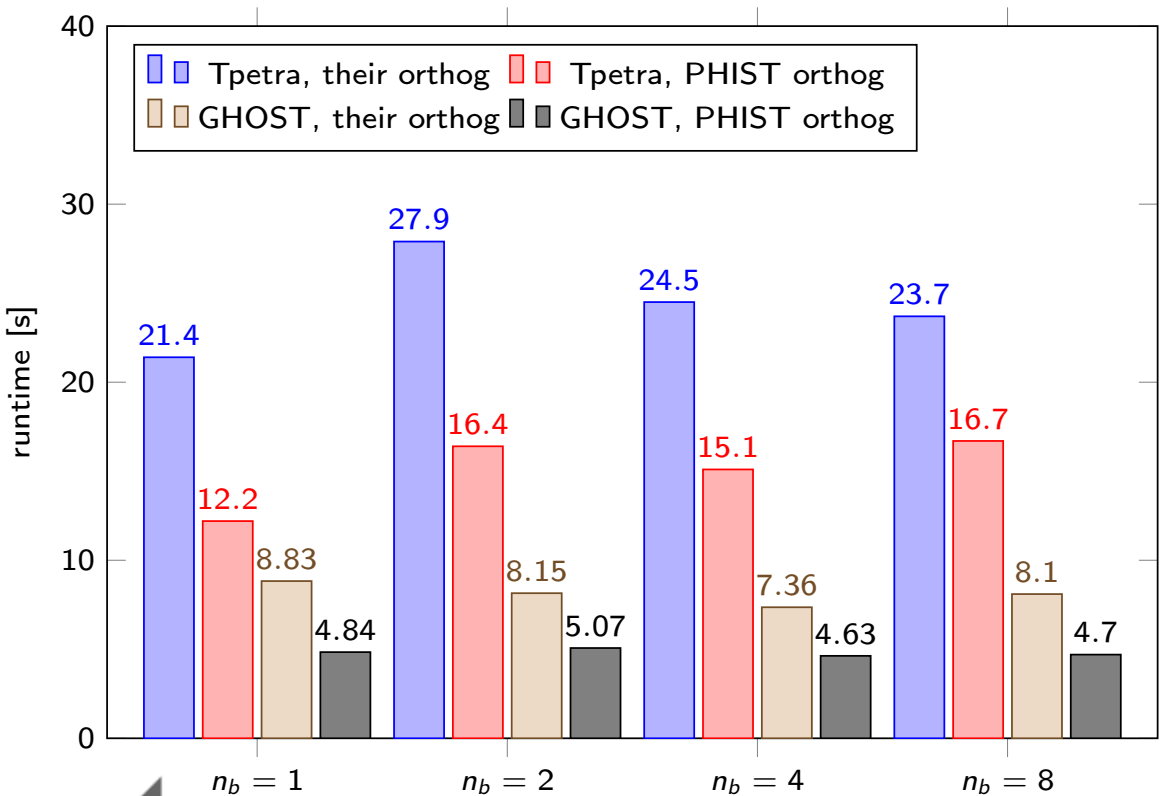
Block Orthog: runtime to convergence



Orthog. n_b vectors against a block of n_{proj}, 12-core Haswell CPU

Example: Anasazi Block Krylov-Schur on Skylake CPU

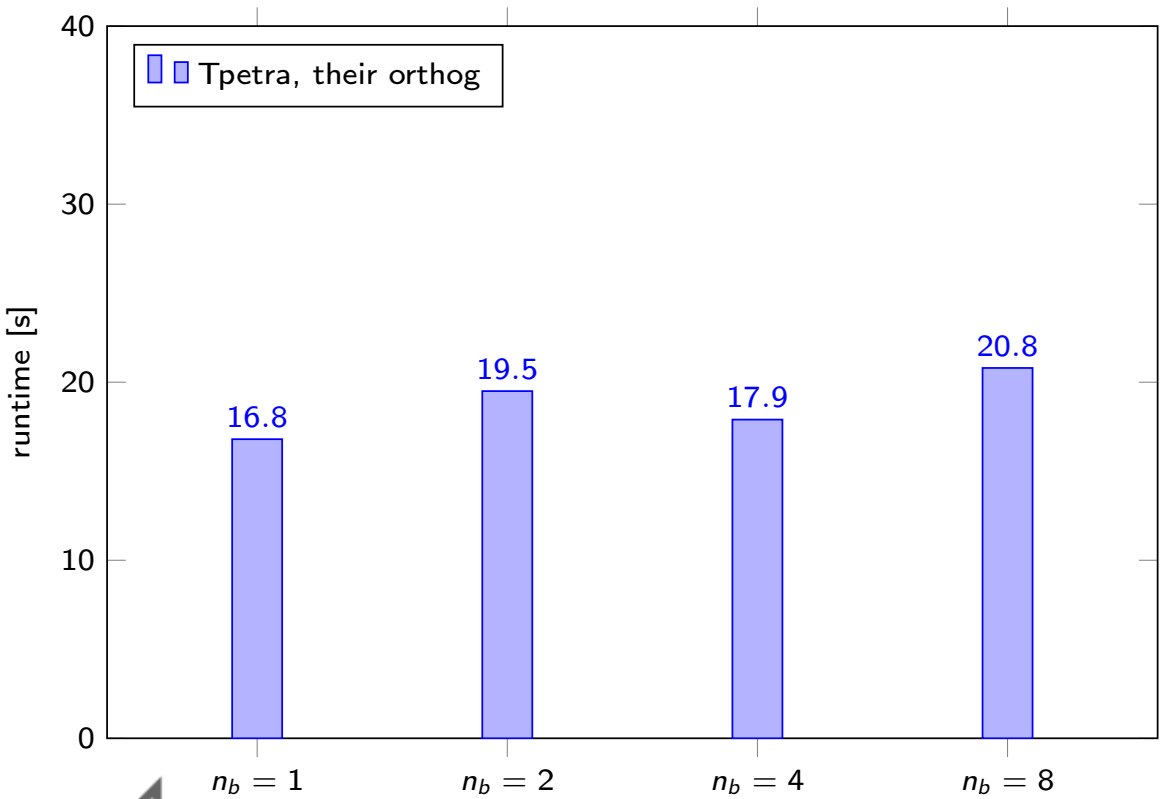
Matrix: non-symmetric 7-point stencil, $N = 128^3$
(var. coeff. reaction/convection/diffusion)



- 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)

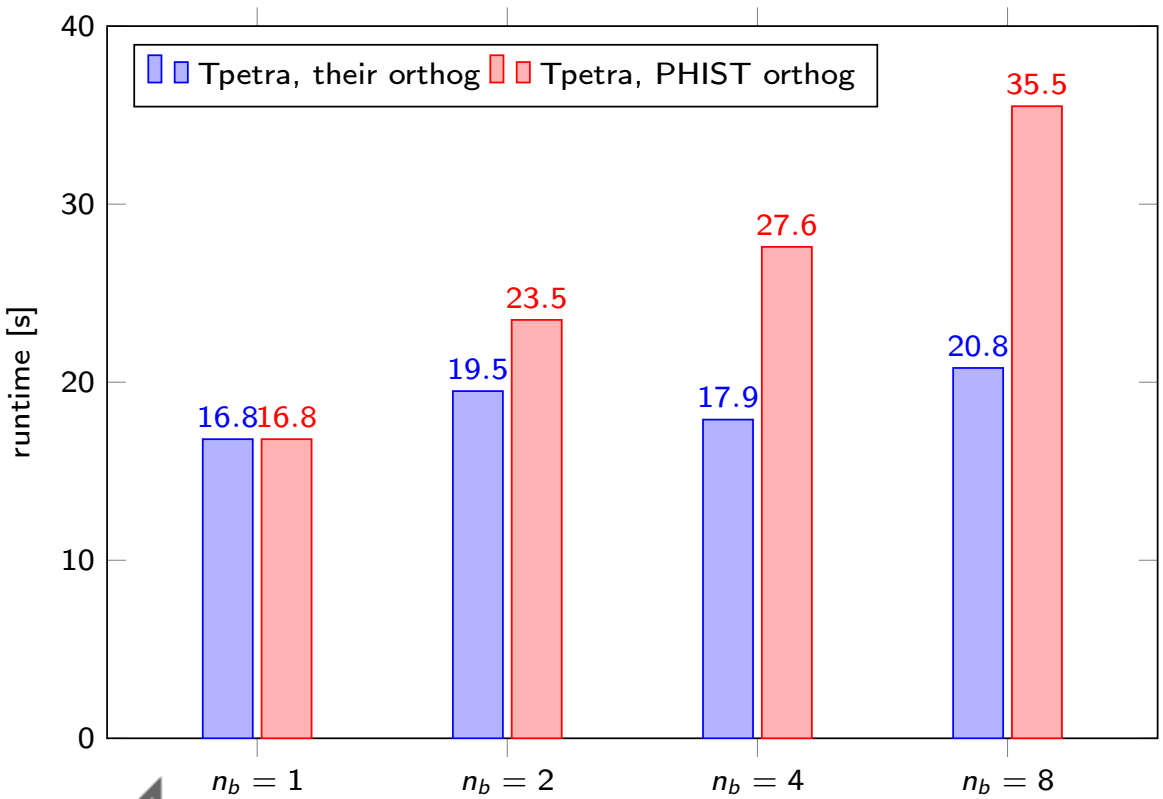


- Anasazi's kernel interface is mostly a subset of PHIST's
  $\Longrightarrow$ extends PHIST by e.g. BKS and LOBPCG
- not optimized for block vectors in row-major storage

DLR

Example: Anasazi Block Krylov-Schur on Volta GPU

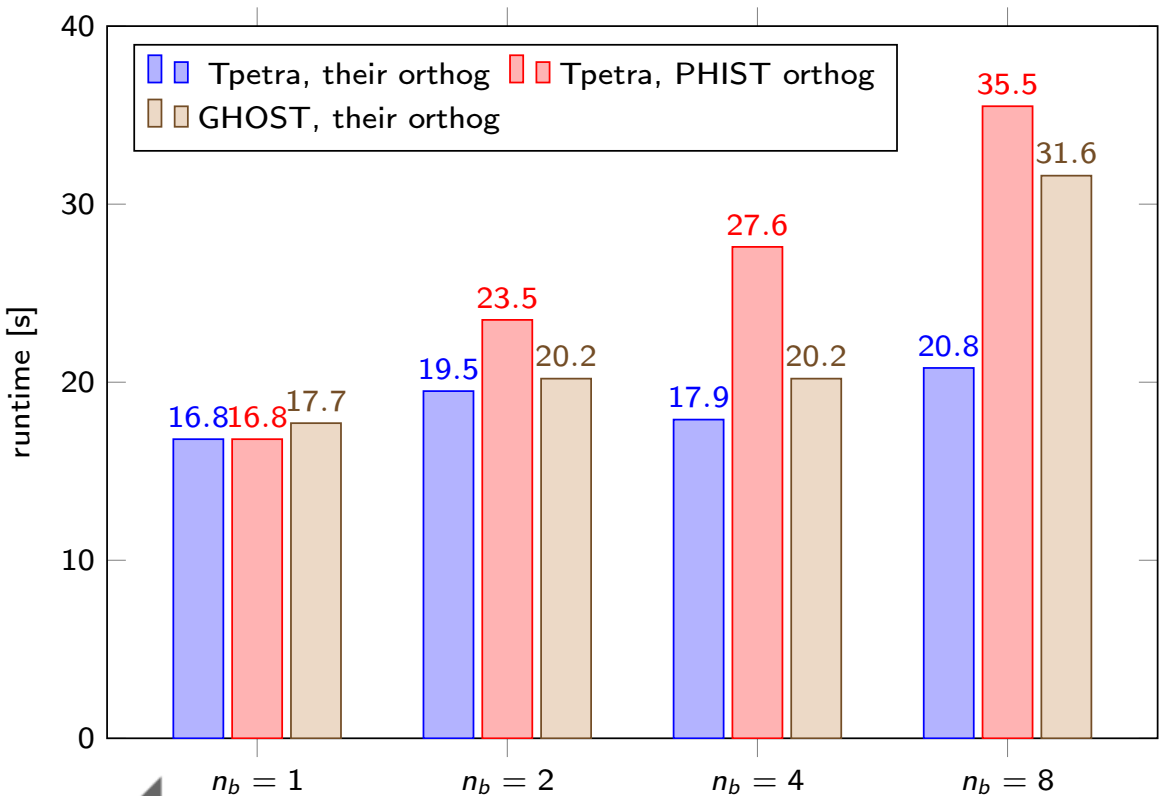
Matrix: non-symmetric 7-point stencil, $N = 128^3$
(var. coeff. reaction/convection/diffusion)



- 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

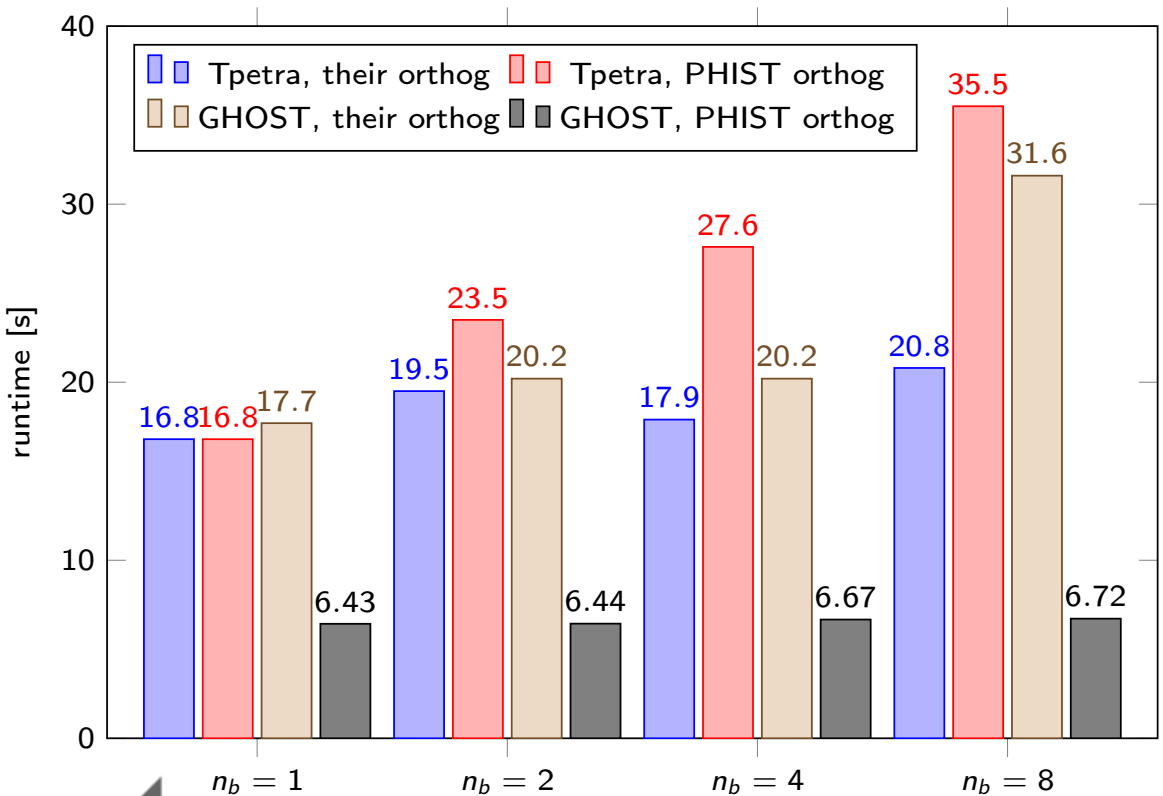
Matrix: non-symmetric 7-point stencil, $N = 128^3$
(var. coeff. reaction/convection/diffusion)



- 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

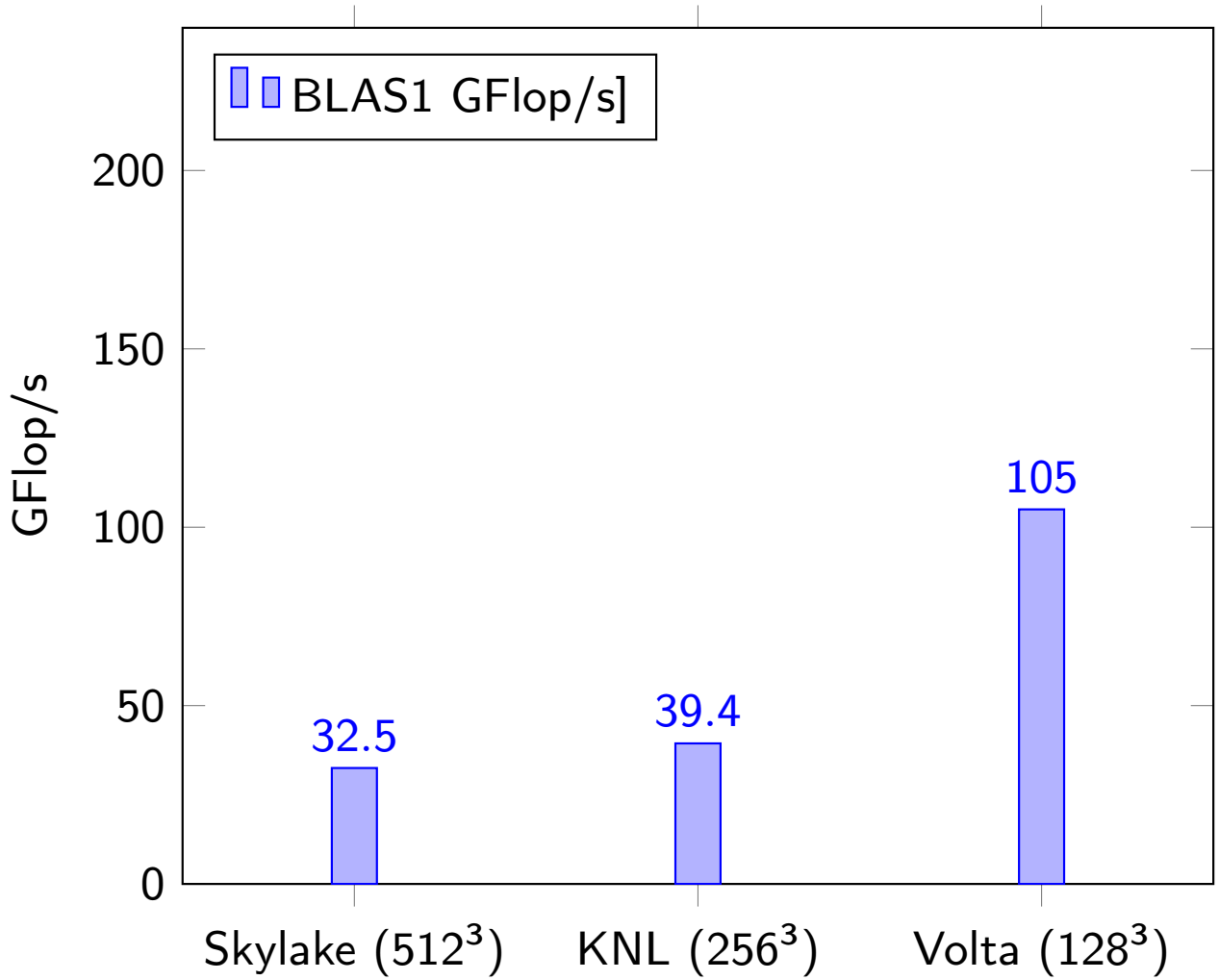
Example: Anasazi Block Krylov-Schur on Volta GPU

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

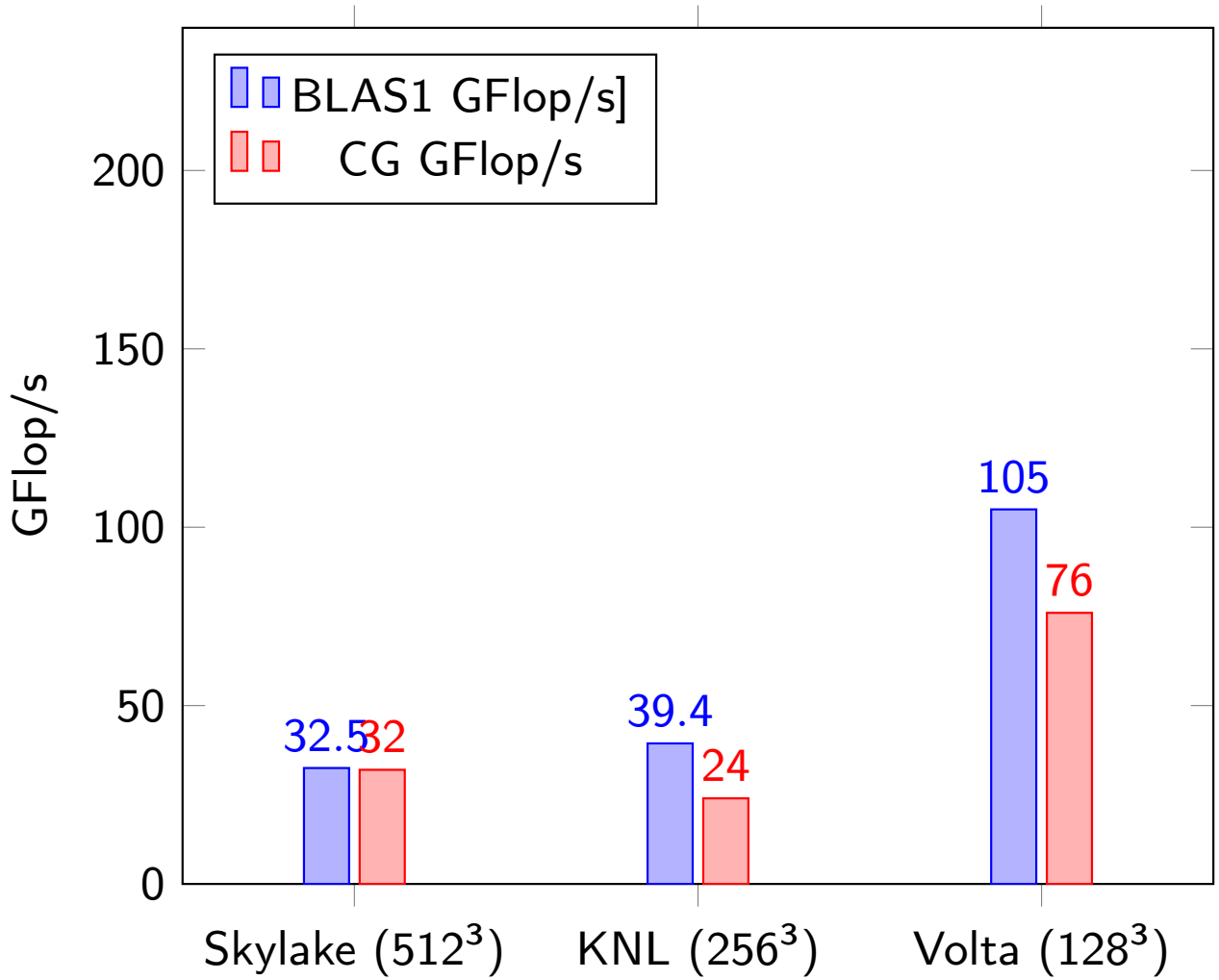


- Anasazi's kernel interface is mostly a subset of PHIST's
  $\Longrightarrow$ extends PHIST by e.g. BKS and LOBPCG
- not optimized for block vectors in row-major storage

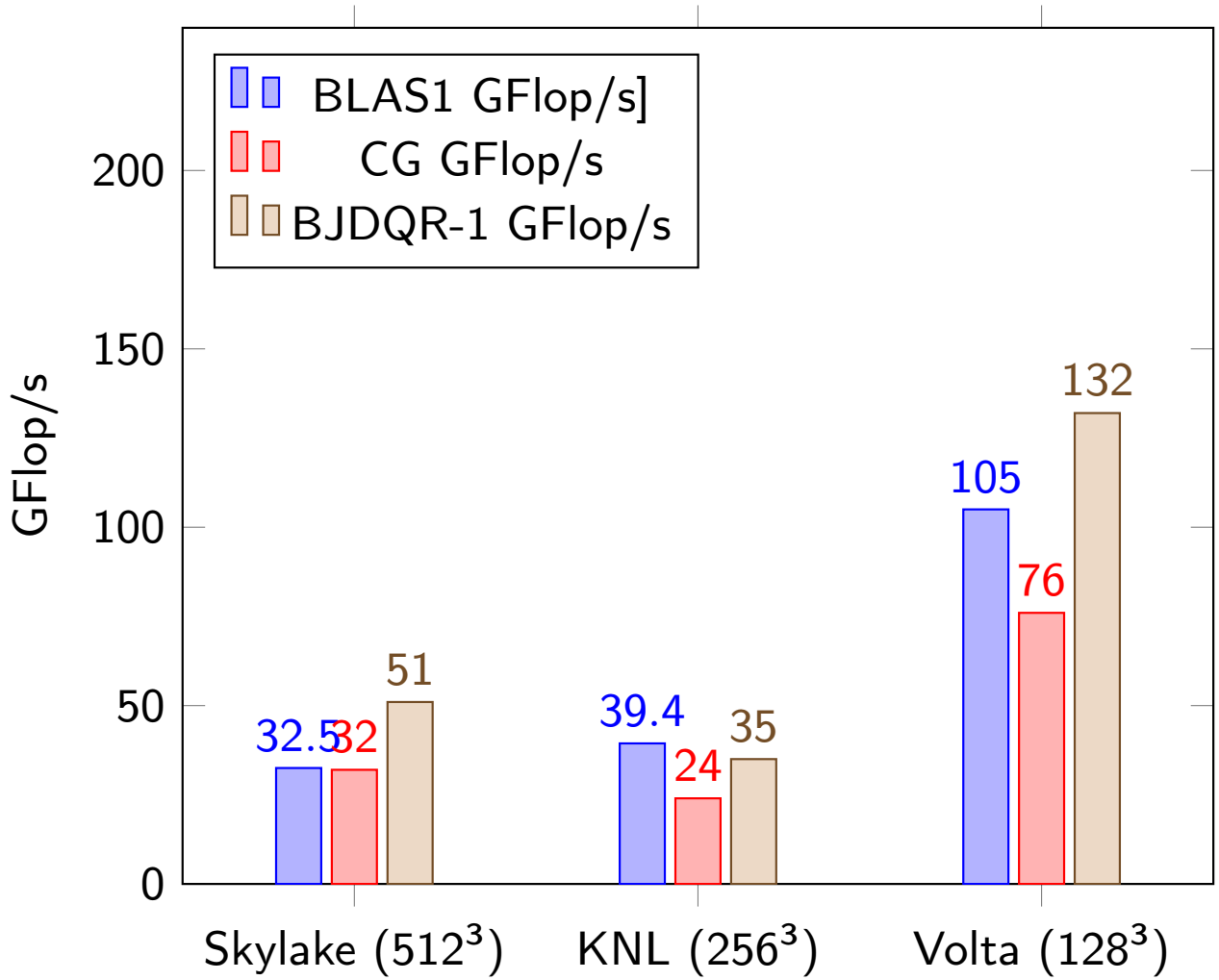
BJDQR on Different Hardware (here: Laplace problem)



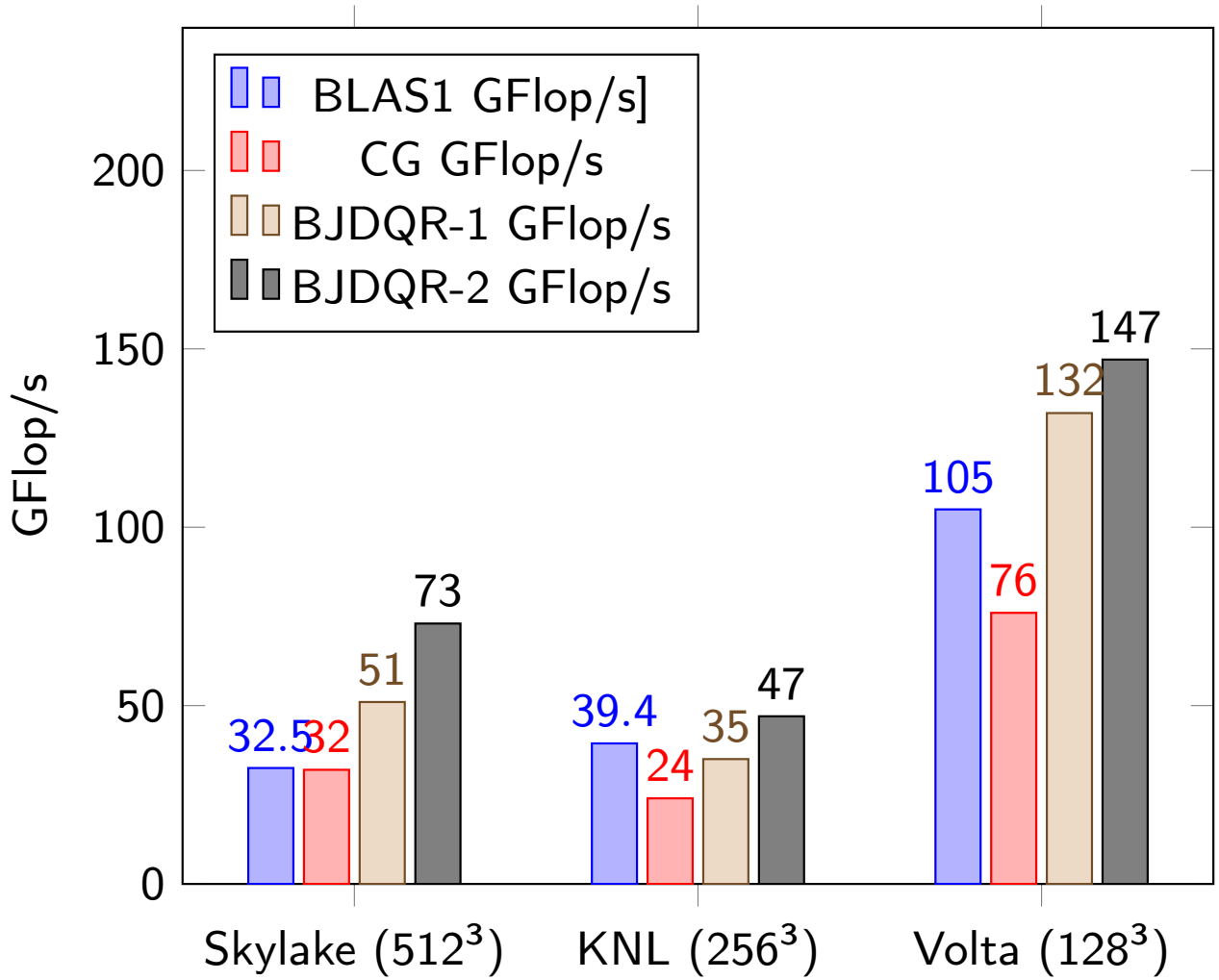
BJDQR on Different Hardware (here: Laplace problem)



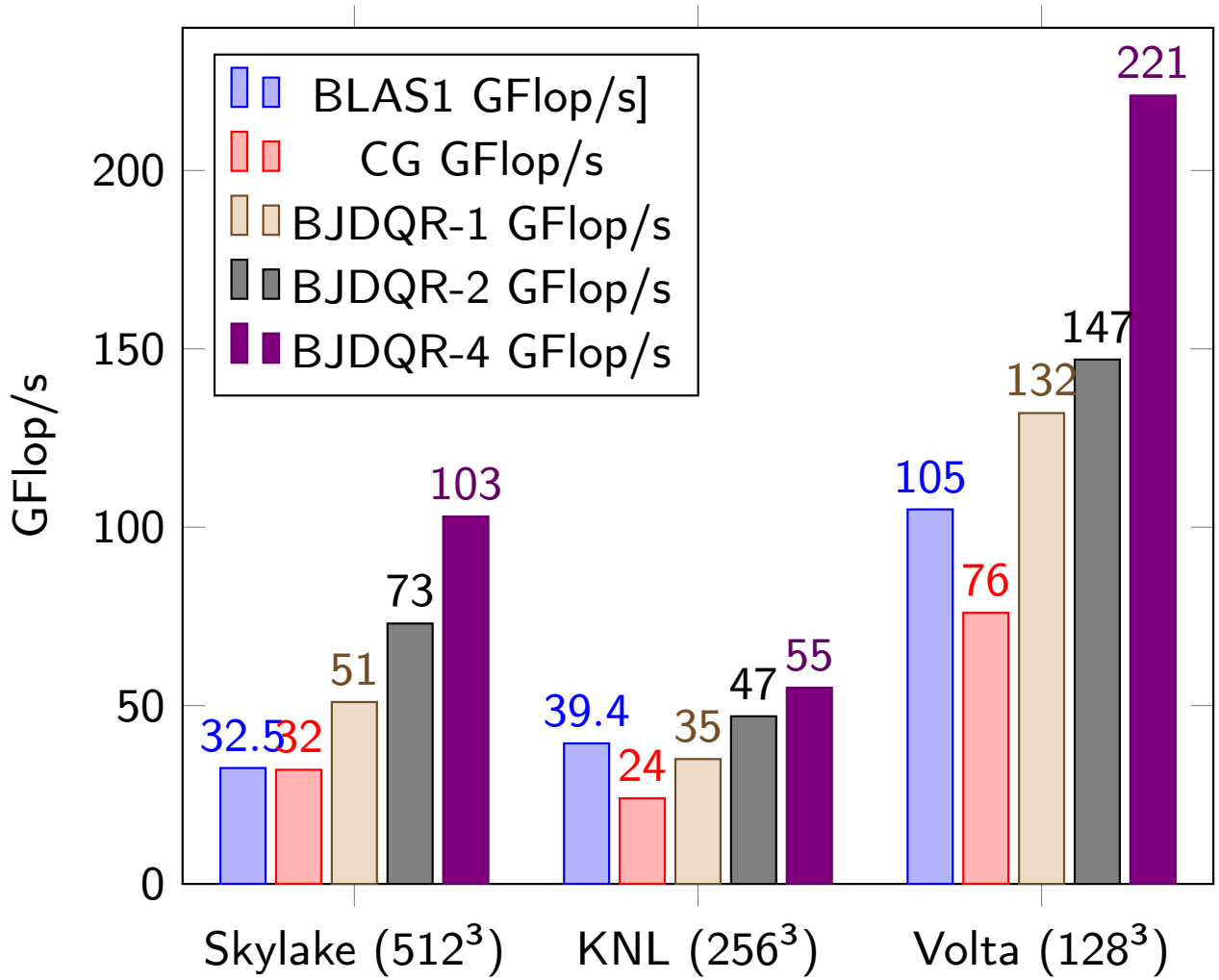
BJDQR on Different Hardware (here: Laplace problem)



BJDQR on Different Hardware (here: Laplace problem)



BJDQR on Different Hardware (here: Laplace problem)



Scaling Experiments

Oakforest-PACS@JCAHPC

Cores:	556 104
Memory:	919 296 GB
Processor:	Intel KNL 68C@1.4GHz
Interconnect:	Intel Omni-Path
Linpack (Rmax)	13 554.6 TFlop/s
Rpeak	24 913.5 TFlop/s
Nmax	9 938 880
HPCG [TFlop/s]	385.479

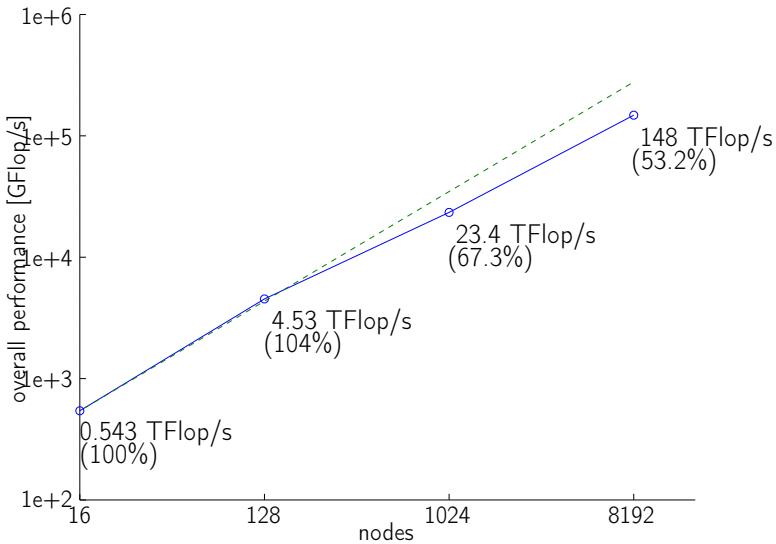
Benchmarks

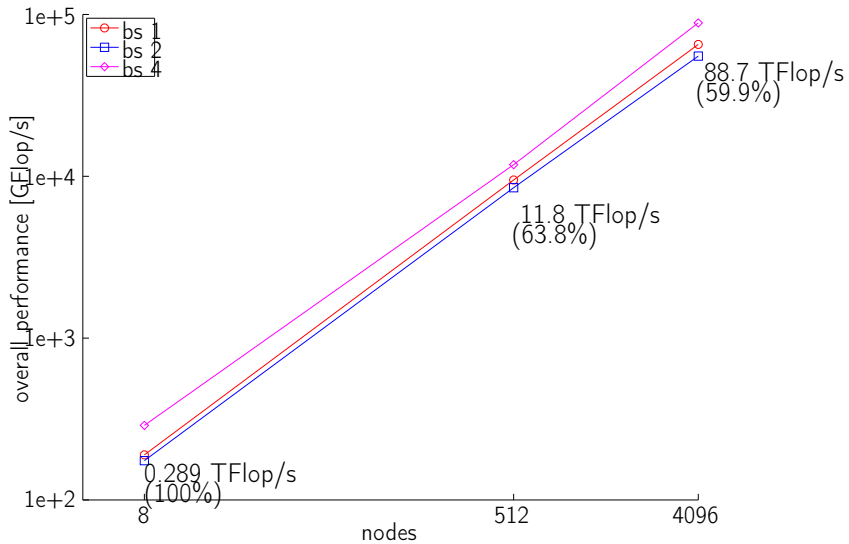
matrices	
symmetric	7-point Laplace, 8.4M rows/node
general	7-point PDE, 2M rows/node
solver parameters	
Krylov solver	10 iterations of MINRES or GMRES+IMGS ortho
JD basis	16-40 vectors
target eigs	near 0 ('SR')

Note:

- fixed number of 250 JDQR iterations
- HPCG uses stencil optimizations
- all data fits in HBM
- experiments not exhaustive

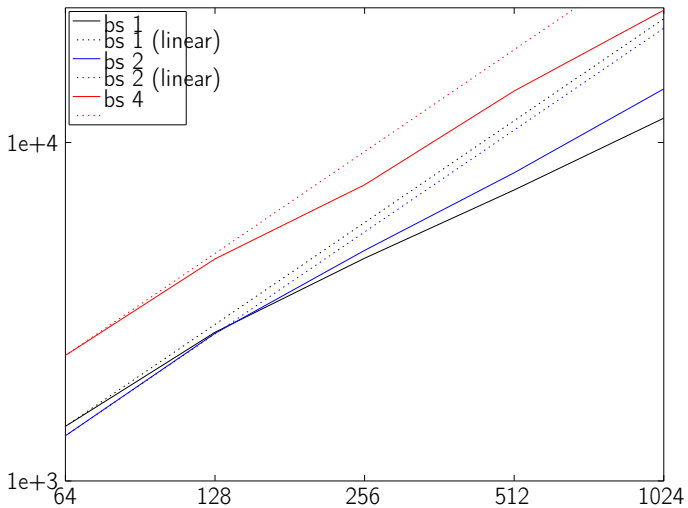
Weak Scaling

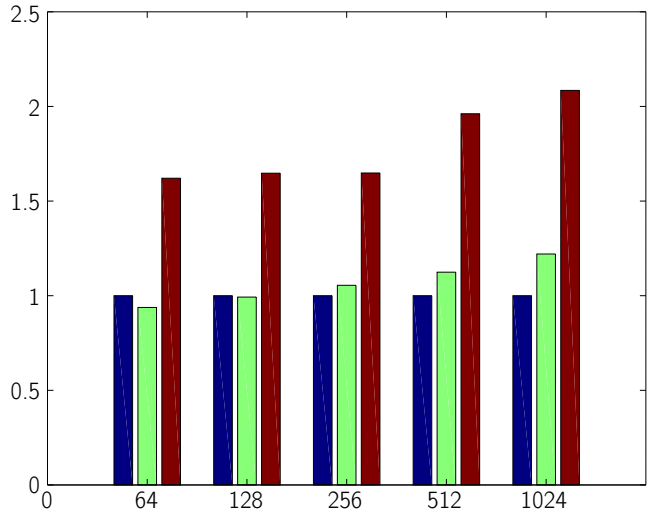




Left: symmetric case up to $N = 4\ 096^3$ (block size 4)
Right non-symmetric case up to $N = 2\ 048^3$

Strong Scaling





Strong scaling for symmetric 1024^3 problem

'block speedup' over $bs = 1$

larger block size...

- increases node-level performance
- decreases number of Allreduce ops
- **but** increases number of iterations

Summary

Software:

- Three libraries for high-performance eigenvalue solvers
- intended for both applications and algorithms R&D

Block algorithms...

- require (near) roofline performance to 'pay off'
- have an advantage in the strong scaling limit
- but require more memory, which is hard for GPUs

TODO-list: GPU performance

- explore UVM, IBM/Volta
- performance modelling
- large scale systems

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

Acknoledgements

- Melven Röhrig-Zöllner and several students worked on PHIST
- GHOST is developed at RRZE (group of Prof. Wellein)
- the ScaMaC is forked in PHIST but developed at U. Greifswald (group of Prof. Fehske)
- ESSEX project: `https://blogs.fau.de/essex/`
- funded by DFG program SPPEXA: `http://www.sppexa.de`
- access to Oakforest-PACS was kindly granted by JCAHPC

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