# The Jacobi-Davidson Eigensolver on GPU ~~Clusters~~

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


project ESSEX

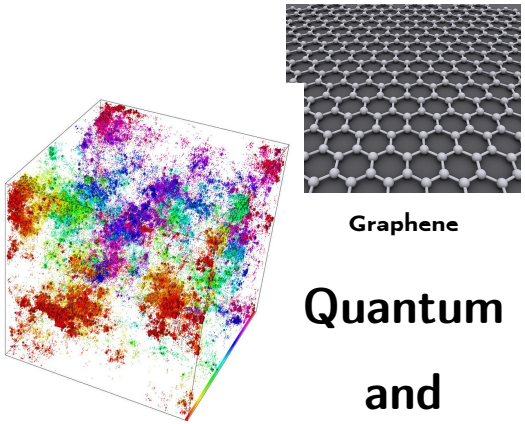

# Sparse Eigenvalue Problems

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

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

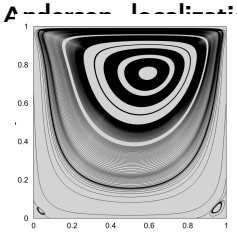
- **A** Hermitian or general, real or complex
- **B** may be identity matrix or Hermitian pos. def.

## Applications

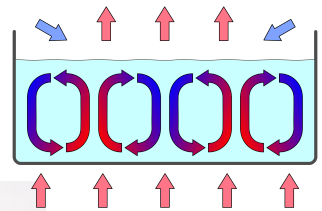



Anderson localization

Hubbard model



Driven cavity



Rayleigh-Benard convection

DLR applications

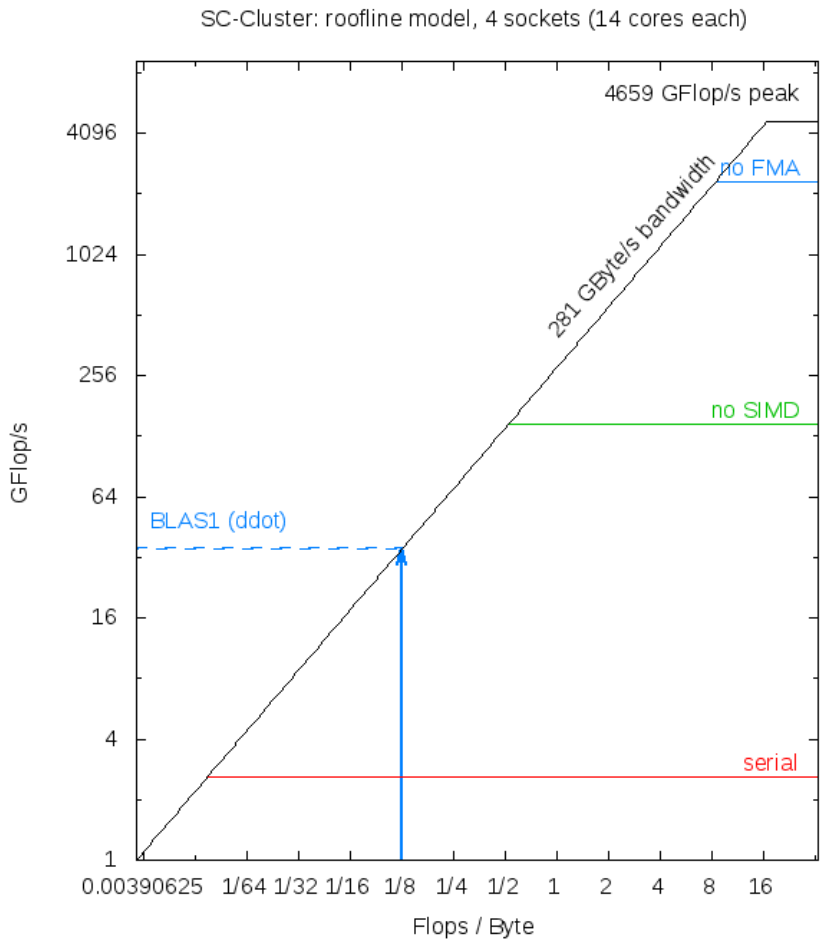
## Performance of Sparse Matrix Algorithms

**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: (L3) cache latency $\sim$ ns
- GPU: launch latency $\sim \mu$s

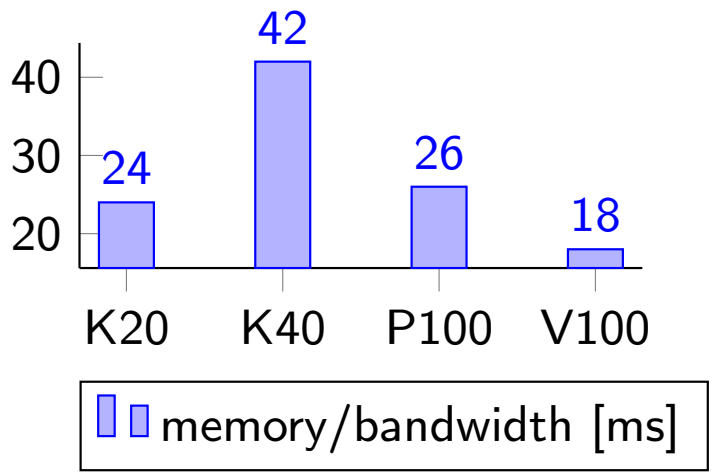


# The Hungry Beast

- "Skylake": Intel Xeon Scalable, 4 × 14 cores @2.6GHz, **384 GB DDR4** RAM
- "Volta": NVidia Tesla V100-SXM2 GPU, **16 GB HBM2** memory

| benchmark | Skylake | Volta |
|---|---|---|
| load | 360 | 812 |
| store | 200 | 883 |
| triad | 260 | 843 |

Measured streaming memory bandwidth [GB/s]




# The Hungry Beast

- "Skylake": Intel Xeon Scalable, 4 × 14 cores @2.6GHz, **384 GB DDR4** RAM
- "Volta": NVidia Tesla V100-SXM2 GPU, **16 GB HBM2** memory

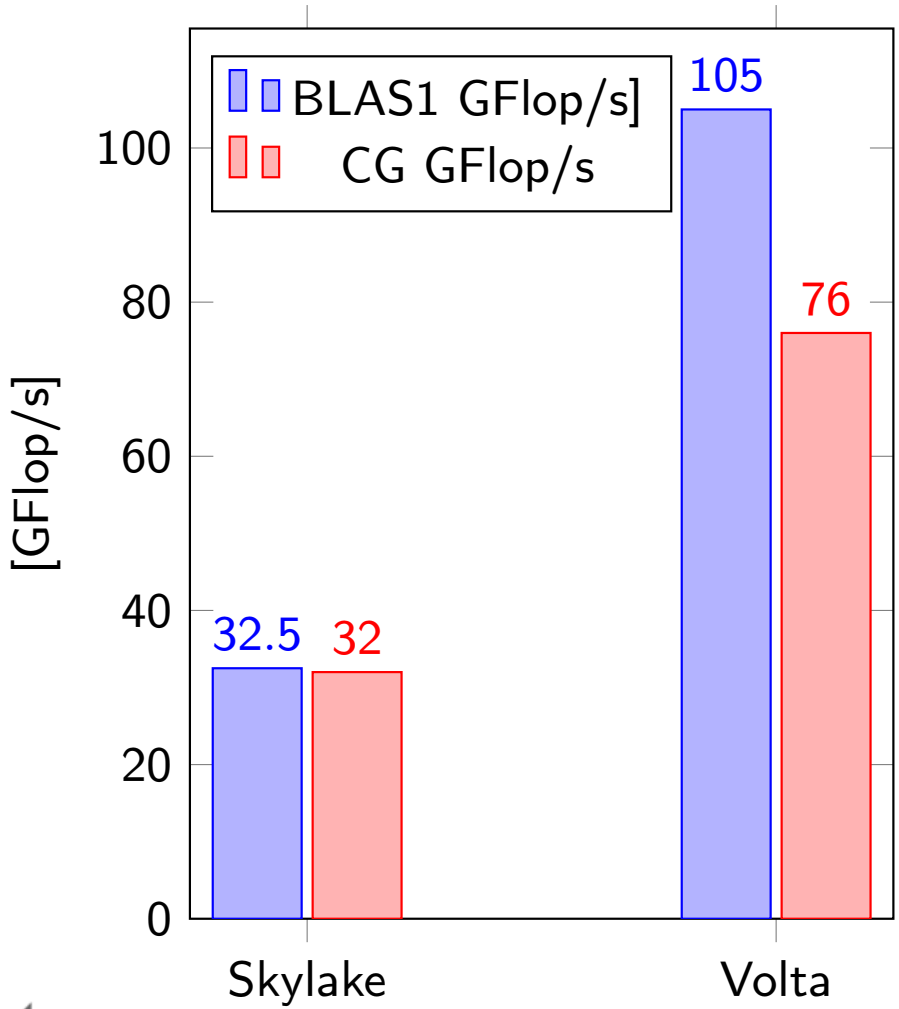
| benchmark | Skylake | Volta |
|---|---|---|
| load | 360 | 812 |
| store | 200 | 883 |
| triad | 260 | 843 |

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 |

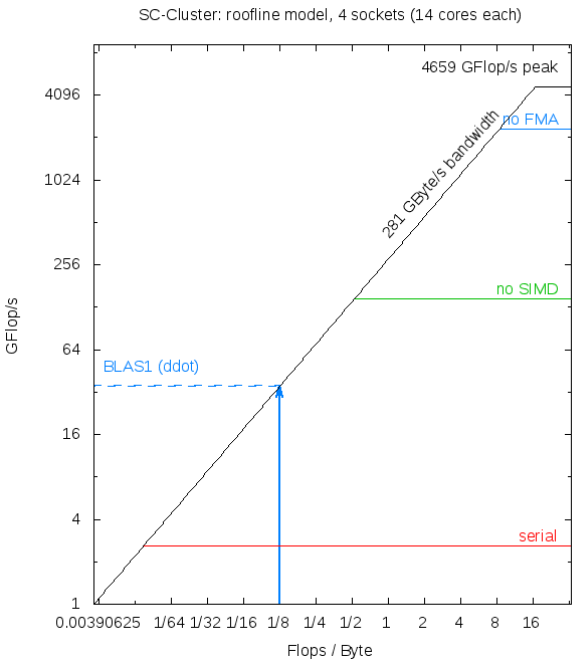
"% roofline" of $X^T Y, X, Y \in \mathbb{R}^{N \times n_b}$ on Volta.

## Example 1: Conjugate Gradients (CG)




- 1000 CG iterations
- 3D 7-point Laplacian
- $256^3$ grid (N=16.7M)
- 2.2GB for matrix and vectors

# Increasing the Flop Intensity

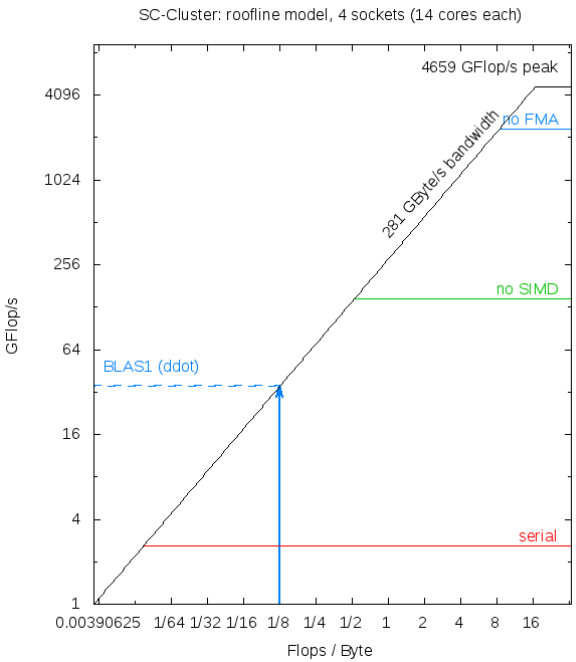



**Block solvers** (block size $n_b$)

- 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 $n_b$)

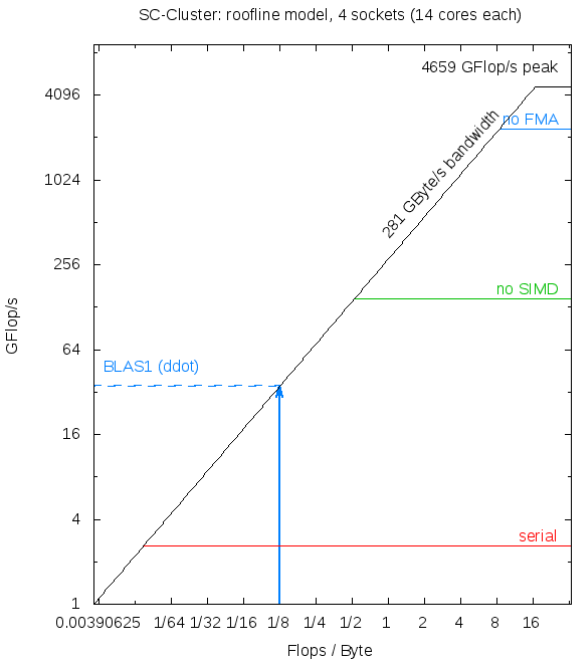
- inner product $\Longrightarrow$ factor $n_b^2$ more flops
- vector updates remain BLAS1 ($X \leftarrow X + \alpha Y$)
- **Caveat:** may increase number of iterations
- **Example:** block GMRES for multiple RHS

**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

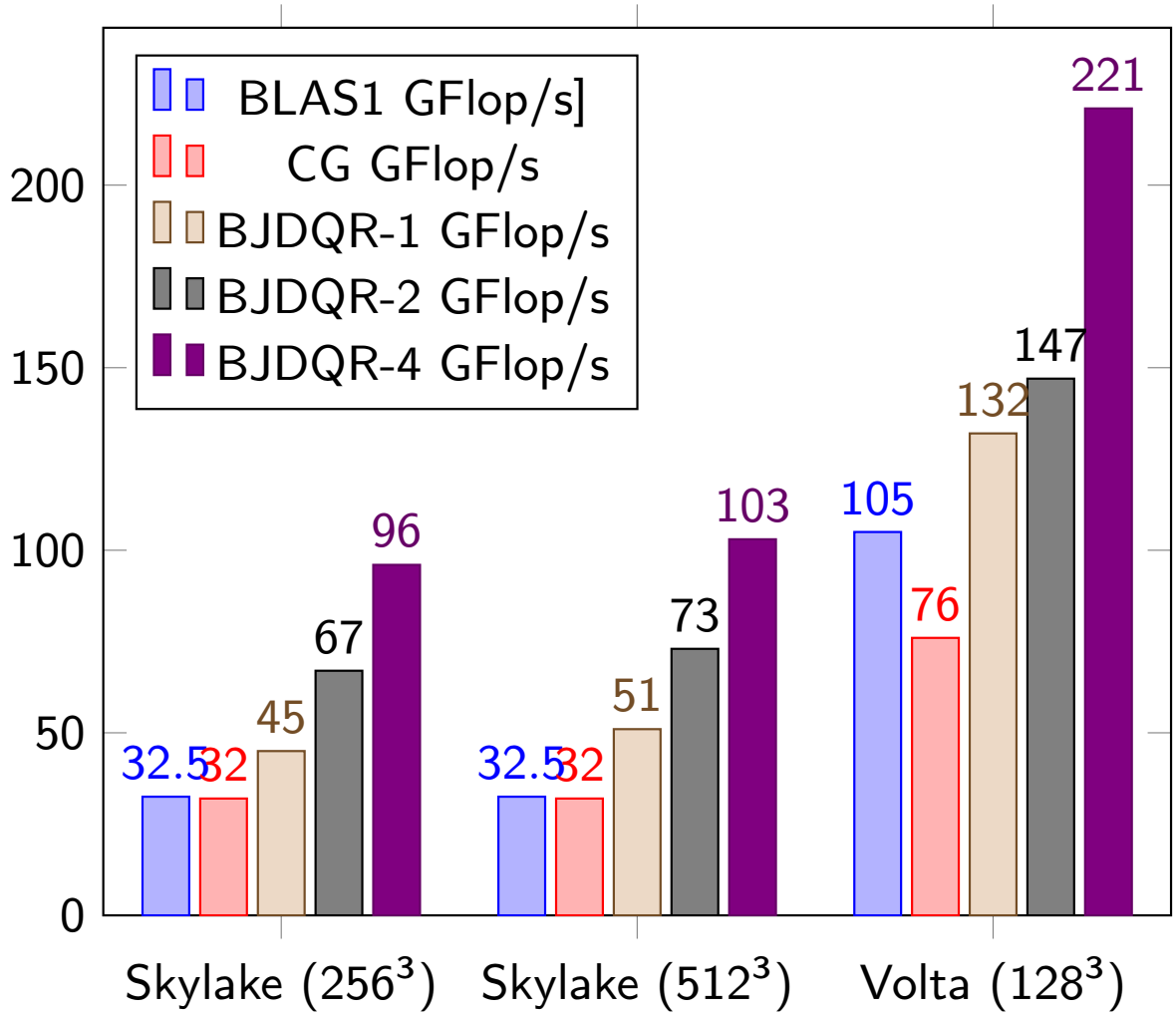
**Kernel fusion**

- 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 2: Block Jacobi-Davidson QR Eigensolver




## Software I: our kernel library

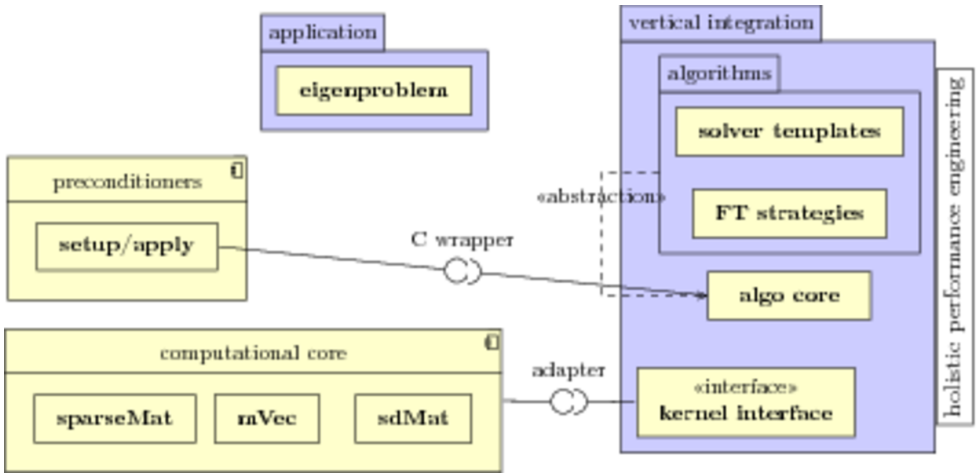
**G**eneral, **H**ybrid-parallel and
**O**ptimized **S**parse **T**oolkit

- 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)
- provides memory-bounded kernels for sparse solvers

## Software II: Algorithms and Interation Framework

# PHIST

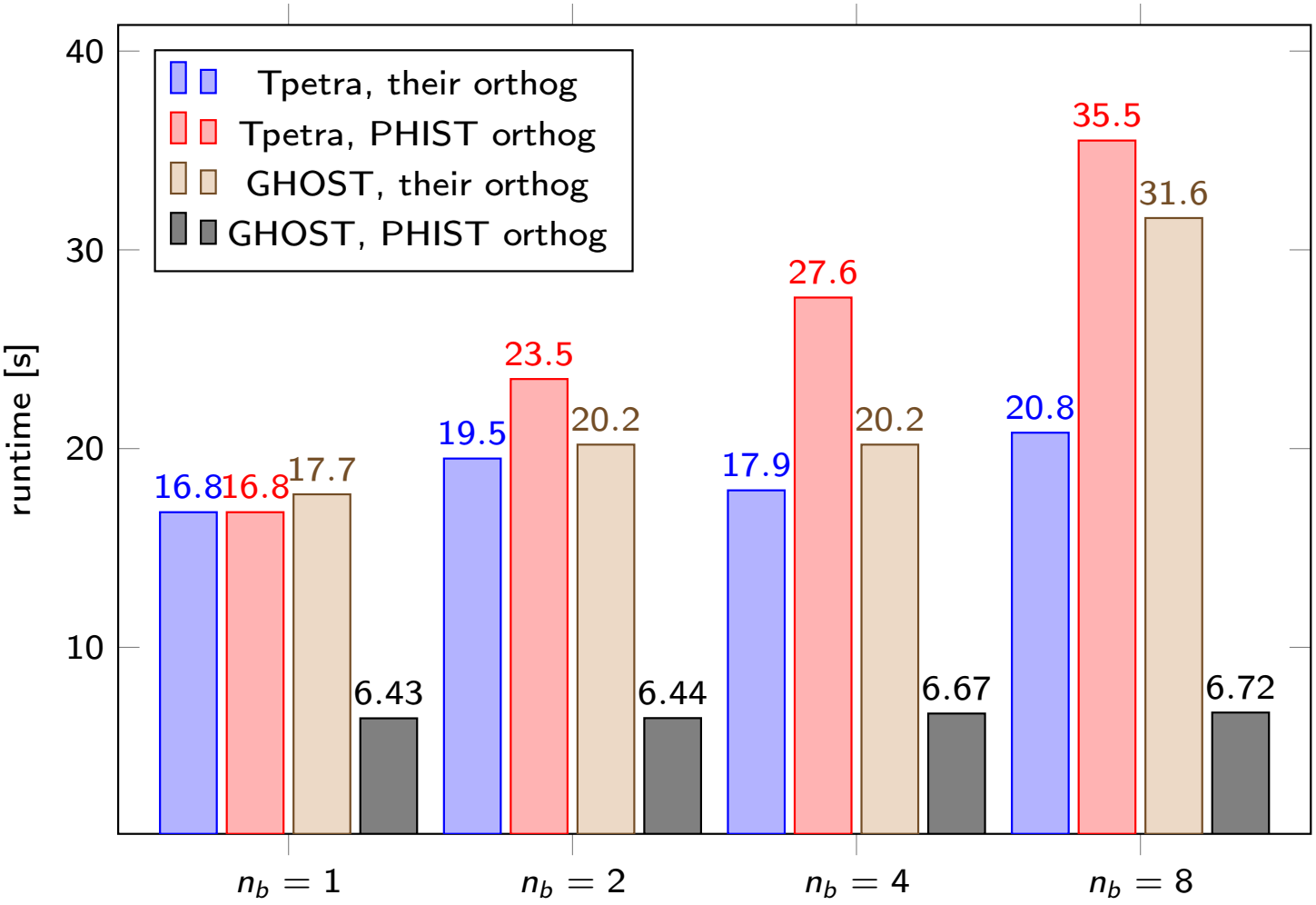
**P**ipelined, **H**ybrid-parallel
**I**terative **S**olver **T**oolkit




- 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

# Example 3: Trilinos Block Krylov-Schur with PHIST+GHOST




## Summary

**How useful are GPUs for sparse Linear Algebra?**

- Roofline performance needs large data sets
- but memory is scarce...
- block algorithms require (near) roofline performance to 'pay off'
- and more memory than their scalar counterparts

**Possible improvements**

- use Unified Virtual Memory (UVM)
- matrix-free applications
- reduced precision vectors

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