The Block Jacobi-Davidson Eigensolver in PHIST

Jonas Thies[1], Melven Röhrig-Zöllner[1], Dominik Ernst[2], Moritz Kreutzer[2], Achim Basermann[1], Georg Hager[2], and Gerhard Wellein[2]

[1]DLR, Simulation and Software Technology,
[2]Erlangen Regional Computing Center

Numerical method

Eigenvalue problem definition

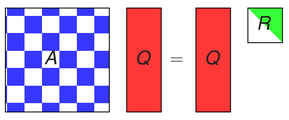
Calculate a small number of extremal eigenpairs (λ_i, v_i) for a sparse, large matrix $A \in \mathbb{R}^{n\times n}$:

$$Av_i = \lambda_i v_i, \qquad i = 1, \ldots, l.$$

With an orthonormal basis $Q = (q_1, \ldots, q_l)$ for the invariant subspace $\mathcal{V} = \mathrm{span}\{v_1, \ldots, v_l\}$ one obtains the more stable block formulation:

$$\begin{cases} AQ - QR & = 0, \\ -\frac{1}{2}Q^*Q + \frac{1}{2}I & = 0. \end{cases}$$

→ Partial Schur decomposition with $r_{i,i} = \lambda_i$:



Block correction equation

In each step of a block Jacobi-Davidson algorithm one calculates correction vectors $\Delta q_1, \ldots, \Delta q_l$:

$$\underbrace{(I - \widetilde{Q}\widetilde{Q}^*)}_{\text{projection onto } Q^\perp} (A - \tilde{\lambda}_i I)(I - \widetilde{Q}\widetilde{Q}^*)\Delta q_i \approx -(A\tilde{q}_i - \widetilde{Q}\tilde{r}_i).$$

- $(\widetilde{Q}, \widetilde{R})$ is the current approximation, $\tilde{\lambda}_i = r_{i,i}$ and $\tilde{r}_i$the ith column of $\widetilde{R}$.
- approximated by some iterations of MINRES or GMRES.

Generalizations

The method is in fact a subspace accelerated Newton-Krylov method applied to multiple eigenpairs (blocking).

Generalizations implemented in PHIST include

- Hermitian or non-Hermitian
- real or complex matrices
- generalized EVP $Ax = \lambda Bx$ for h.p.d. B
- arbitrary preconditioning for the correction equation

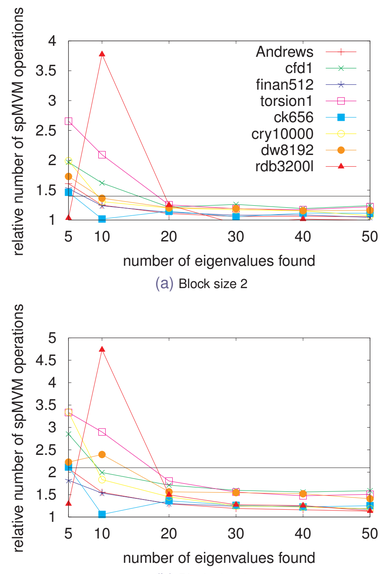
Additional operations due to blocking

- Blocking increases the number of operations (but blocked operations are faster).

→ Question: How large is the overhead?

- Approach to estimate possible performance gains:
  - Count sparse matrix-vector multiplications (spMVM).
  - Relate results to the performance of block spMVMs.

→ For more than 20 eigenpairs blocking may be beneficial.



(a) Block size 2

(b) Block size 4

Figure 1: Number of spMVMs of block JDQR compared to single-vector JDQR.

Preconditioning

If K^{-1} is a suitable preconditioner for $A - \tau B$ for some τ near the sought eigenvalues, left preconditioning is implemented by a 'skew-projected' preconditioning operator:

$$\mathrm{precOp}_{\mathrm{B}} = (I - (K^{-1}\widetilde{Q})((B\widetilde{Q})^H K^{-1}\widetilde{Q})^+(B\widetilde{Q})^H)K^{-1}.$$

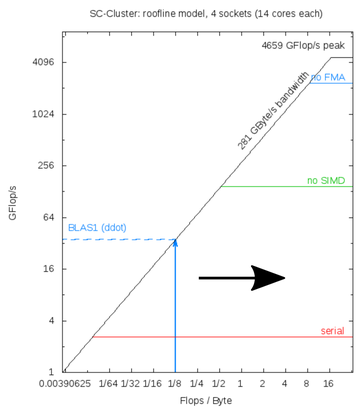
The projection makes sure that the 'inner' Krylov space stays orthogonal to the current approximation and locked eigenvectors in $\widetilde{Q}$.

Table 1: Effect of using an AMG preconditioner with the Jacobi-Davidson method. 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

Note: preconditioner performance should benefit from block-speedup similar to spMVM, not investigated further here.

Node-level performance



- Node-level performnance of all kernels is bounded by the memory bandwidth
- Can only get faster by doing more Flops/Byte
- 'tall and skinny' kernels: e.g. $M \leftarrow X^TY, X \leftarrow X \cdot M; X, Y \in \mathbb{R}^{N\times k}, M \in \mathbb{R}^{k\times k}$
- PHIST allows printing kernel 'roofline performance' after run

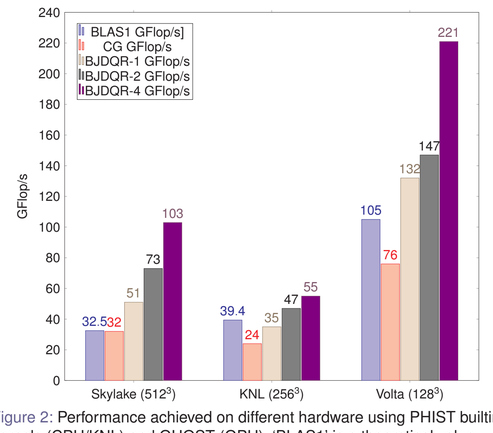


Figure 2: Performance achieved on different hardware using PHIST builtin kernels (CPU/KNL) and GHOST (GPU). 'BLAS1' is a theoretical value (ddot according to memory bandwidth).

Why is the GPU not as fast as expected?

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

Table 2: Measured streaming memory bandwidth [GB/s]

GPUs are increasingly hungry

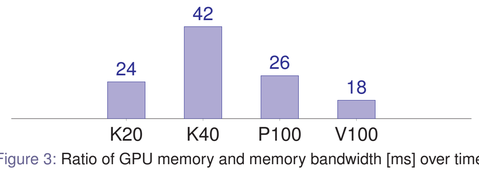


Figure 3: Ratio of GPU memory and memory bandwidth [ms] over time.

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

Table 3: "% roofline" of $X^TY, X, Y \in \mathbb{R}^{N\times n_b}$ using GHOST on Volta. Optimal performance is requires block vectors of at least 1*GB* each!

GHOST now has experimental support for CUDA UVM, but the performance assessment is tricky.

Scalability on Oakforest-PACS

Machine

Figure 4: Impression of the Oakforest-PACS supercommputer at the Japanese joint center for advanced HPC (JCAHPC).
http://jcahpc.jp/ofp/ofp_intro.html

Cores:	556,104
Memory:	919,296 GB
Processor:	Intel Xeon Phi 7250 68C 1.4GHz
Interconnect:	Intel Omni-Path
Linpack Performance (Rmax)	13,554.6 TFlop/s
Theoretical Peak (Rpeak)	24,913.5 TFlop/s
Nmax	9,938,880
HPCG [TFlop/s]	385.479

Table 4: System specification of Oakforest-PACS.
https://top500.org/

Benchmarks

matrices	
symmetric	7-point Laplace, 8.4M rows/node
general	7-point, some PDE, 2.0M rows/node
solver parameters	
Krylov solver	10 iterations of MINRES (sym.) or GMRES+IMGS ortho (general)
JD basis	16-40 vectors
target eigenpairs	near 0 ('SR')

Note:

- We ran the solver for a fixed number of 250 Jacobi-Davidson iterations because convergence depends strongly on problem size without additional preconditioning.
- We do not get near the HPCG performance in our experiments because we do not exploit the matrix structure. Furthermore, the problem sizes are rather small to fit all the required vector spaces into the high-bandwidth memory (HBM).
- Due to limited CPU time we could only run a few benchmarks and some data points are missing in the series.

Weak scaling

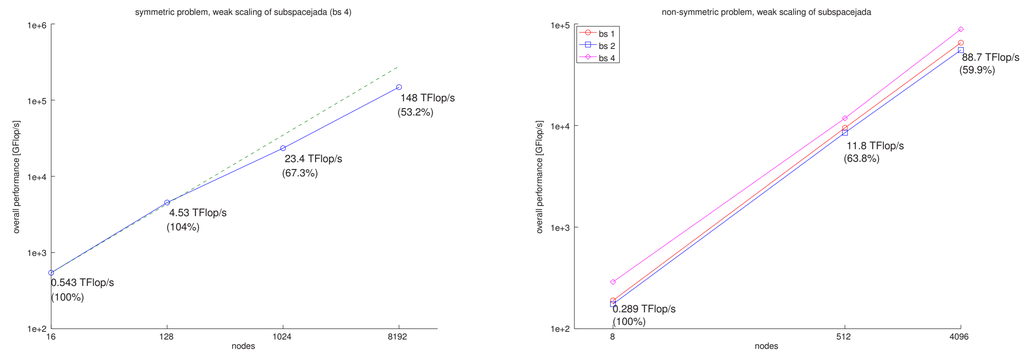


Figure 5: Weak scaling on up to 0.5M cores. The percentage indicates the parallel efficiency compared to the first measurement (smallest node count). Left: symmetric PDE problem with the largest matrix size $N = 4\,096^3$, right non-symmetric PDE problem with the largest problem size $N = 2\,048^3$. The best performance was obtained with a block size of 4. The numbers in the plot refer to this case.

Strong scaling

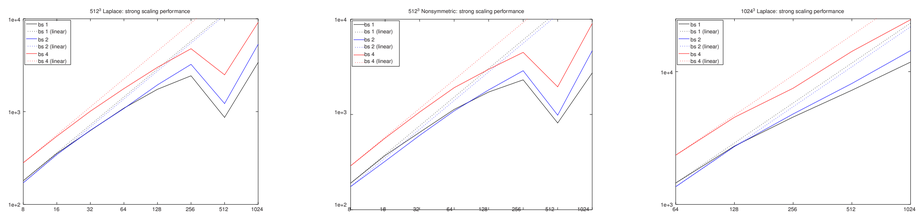


Figure 6: Strong scaling: larger block size reduces number of Allreduce operations. The performance drop at 512 nodes and $N = 512^3$ may need further investigation.

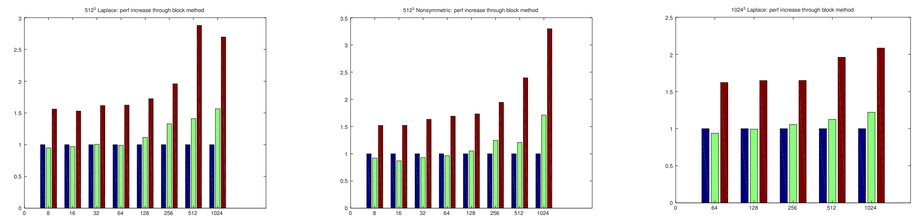


Figure 7: strong scaling: corresponding 'block speedup' over the bs=1 case. The KNL doesn't seem to 'like' block size 2 very much (in contrast to Xeon CPUs). Maybe the bandwidth can't be saturated with SSE?

Software

Our Hybrid-parallel kernel library

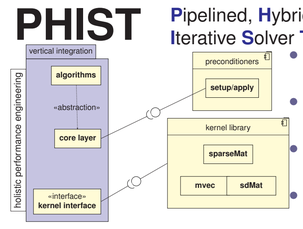
GHOST — **G**eneral, **H**ybrid-parallel and **O**ptimized **S**parse **T**oolkit

- 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

Algorithms and integration 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 kernel library at compile time:
GHOST, builtin (Fortran), Trilinos, PETSc

Will be in the fall release of the xSDK (https://xsdk.info), a collection of extreme-scale simulation software

https://bitbucket.org/essex/phist

Both libraries also available via Spack (https://spack.io).