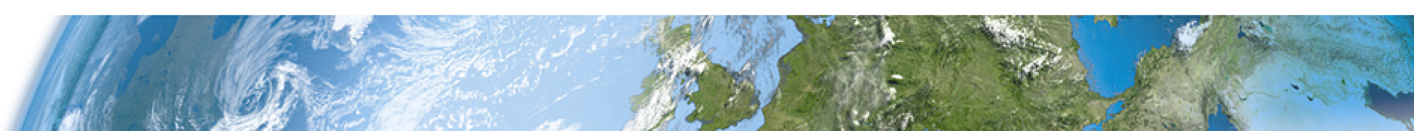
A Hybrid Parallel Block Jacobi-Davidson Method

Jonas Thies, Melven Zöllner, Achim Basermann
Moritz Kreutzer, Faisal Shahzad, Georg Hager, Gerhard Wellein
Andreas Alvermann, Andreas Pieper, Holger Fehske

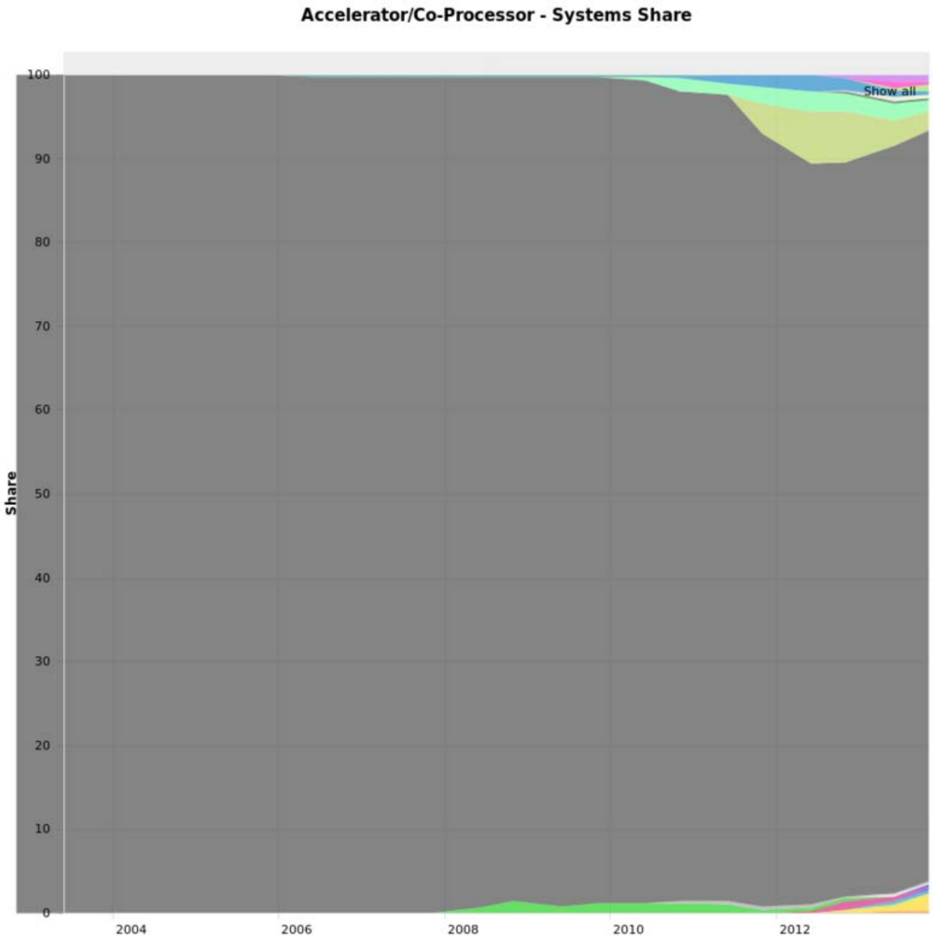
Motivation: why do we need exascale computers in quantum mechanics?

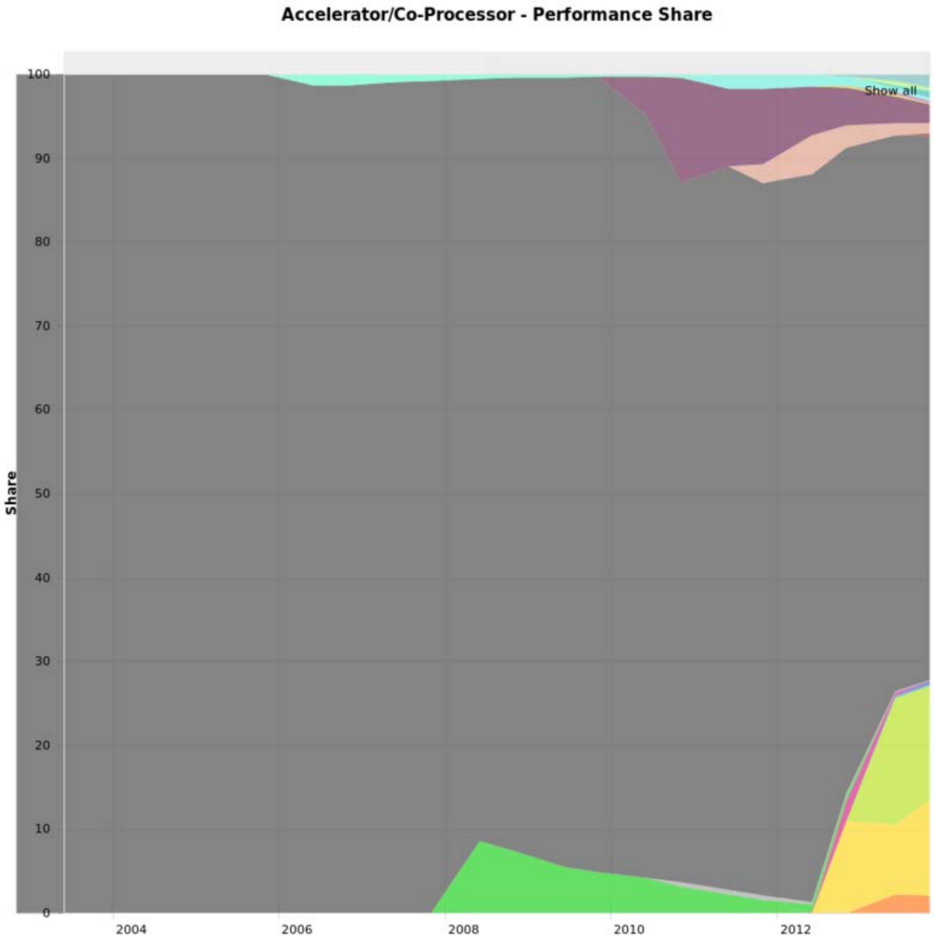
- *L* electrons in a magnetic field
- Each electron has ‘spin up’ (1) or ‘spin down’ (0)
- Superposition of states: vector Ψ of length $N = 2^L$
- Schrödinger Equation:
  - $\boldsymbol{H}\Psi = E\Psi$
- Hamiltonian **H** describes the interaction of neighboring particles

- Possible additional level of parallelism: statistics over randomly perturbed (“disordered”) matrices



Expected target architectures





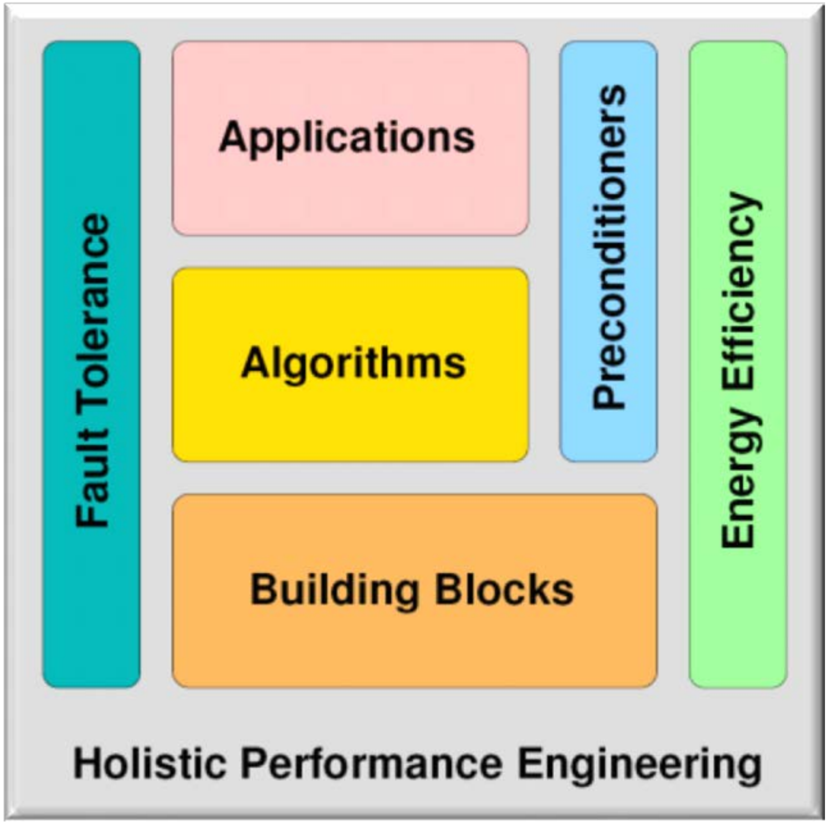
Equipping Sparse Solvers for the EXascale

Presumptions

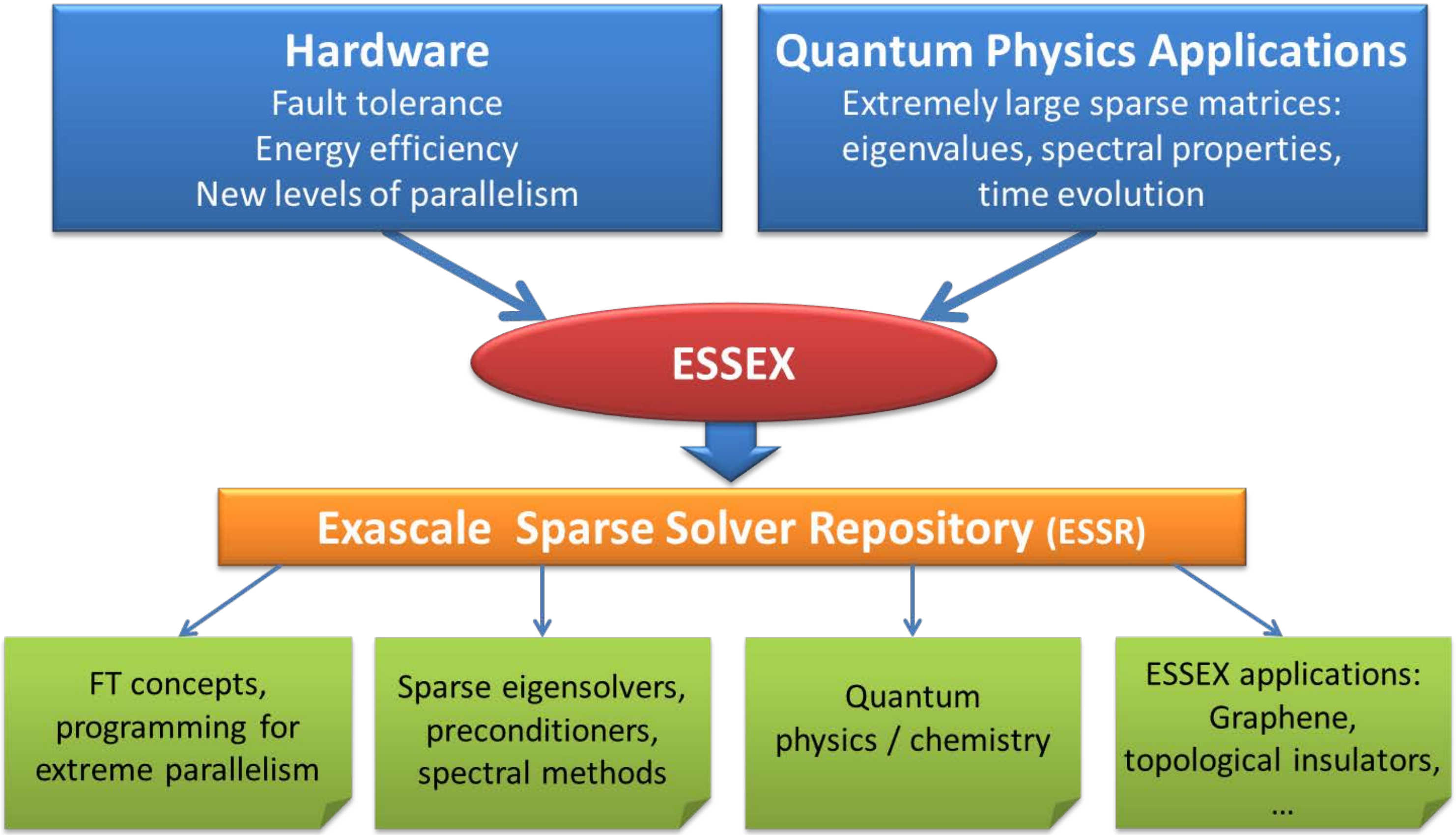
- Heterogenous compute node replaces ‘sequential’
  => MPI+X programming model

- Optimal node level performance is key to energy efficiency and scaling

- Fast hardware deserves fast algorithms



The ESSEX project in a nutshell



Block Jacobi-Davidson QR

- Aim: partial Schur decomposition

$$AQ - QR = 0, \qquad R \in \mathbb{C}^{k\times k} \text{ upper triangular}$$

$$\frac{1}{2}Q^T Q - \frac{1}{2} = 0, \qquad Q \in \mathbb{C}^{N\times k}$$

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

$$A\Delta Q - \Delta Q\tilde{R} \approx \tilde{Q}\tilde{R} - A\tilde{Q},$$

$$\tilde{\mathrm{Q}}^T \Delta Q \approx 0,$$

Block Jacobi-Davidson (2)

- This leads to a correction equation

$$\left(I-\tilde{Q}\tilde{Q}^H\right)A\left(I-\tilde{Q}\tilde{Q}^H\right)\Delta Q-\left(I-\tilde{Q}\tilde{Q}^H\right)\Delta Q\tilde{R} = -(A\tilde{Q}-\tilde{Q}\tilde{R}) \qquad (1)$$

- Subspace acceleration: add search directions to basis *V*
- Ritz-Galerkin: $M = V^H AV,\ M = S^H RS$,
- Restart: shrink basis when it becomes too large
- Locking vs. deflation of converged eigenpairs

DLR

Solving the correction equation

- Eq. (1) a little more readable: find $\Delta Q \in \tilde{Q}^{\perp}$

$$A\Delta Q - \Delta Q\tilde{R} = -\mathrm{res}$$

This is an $N \times k$ dimensional linear system

Replace $\tilde{R}$ by its diagonal
=> decoupled systems
=> still local quadratic (cubic) convergence per eigenvalue, but no longer to the entire subspace

Iterative solution: Krylov method, possibly with preconditioner *P*
Operators $I - \tilde{Q}\tilde{Q}^H$, A and P applied to *k* vectors at a time

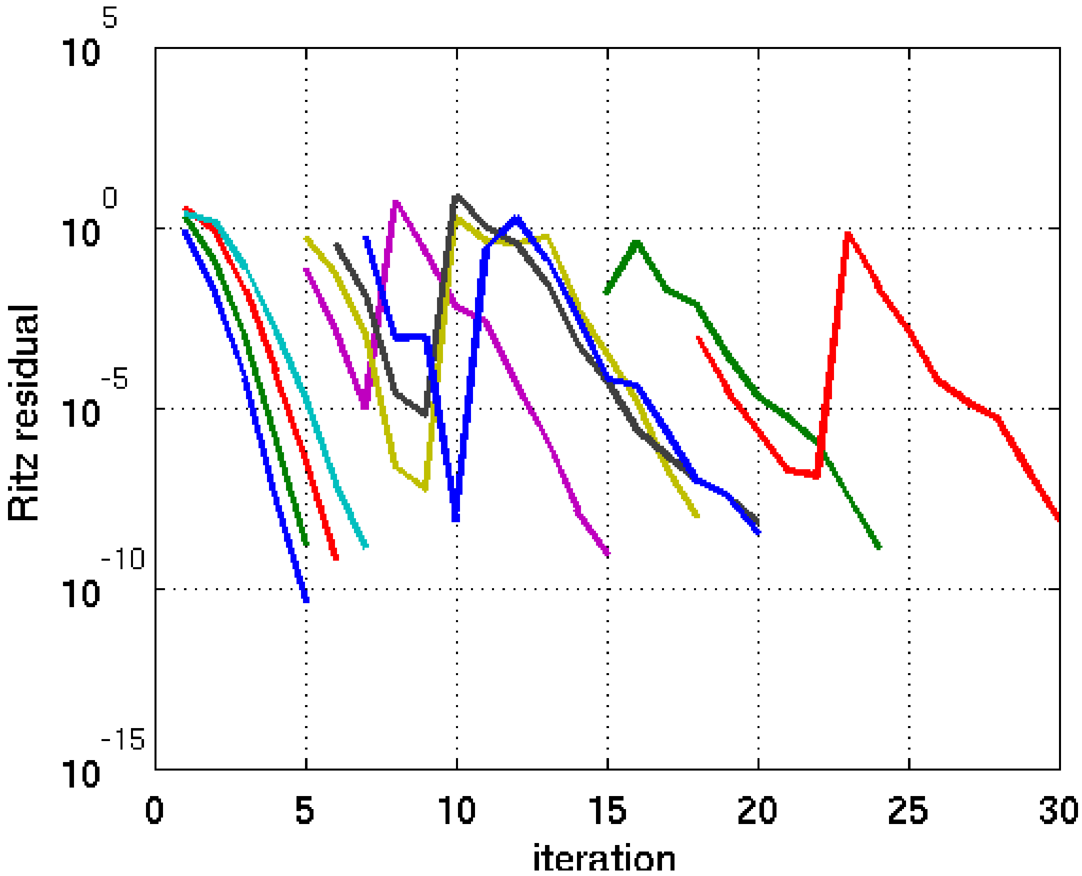
Core operations in block JD

- Sparse Matrix times *k* vectors, $Y = AX$
  - matrix entries loaded into cache once per k vectors (temporal cache locality)
  - communication of *X* in a single message (lower latency penalty)

- Block Gram-Schmidt: $W = W - V(V^T W)$
  - BLAS 3, single message for $(V^T W)$

- Block orthogonalization, $W = QR, Q^T Q = I$
  - TSQR (Hoemmen et al)
    tree algorithm, “communication optimal”
    rank revealing

Typical numerical behavior for fixed block size

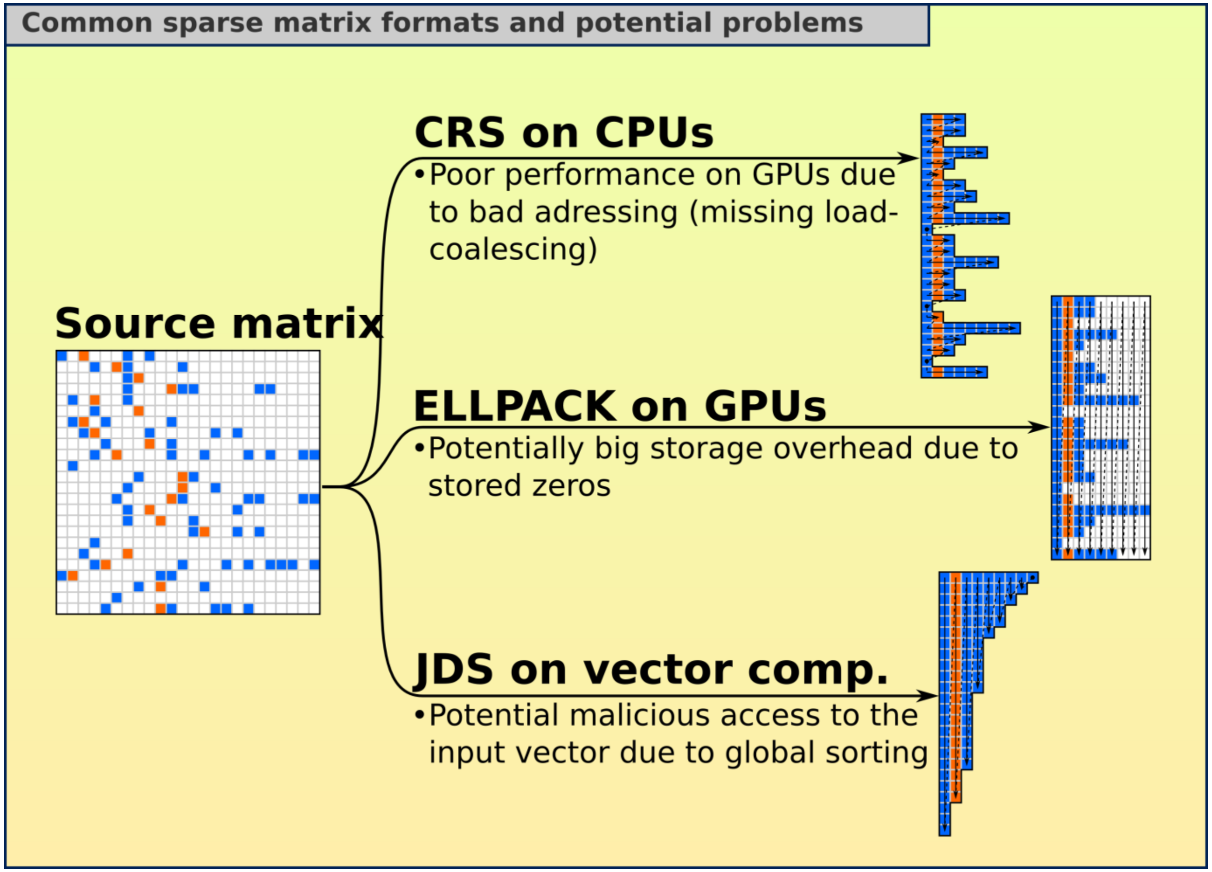


Behavior for increasing block size

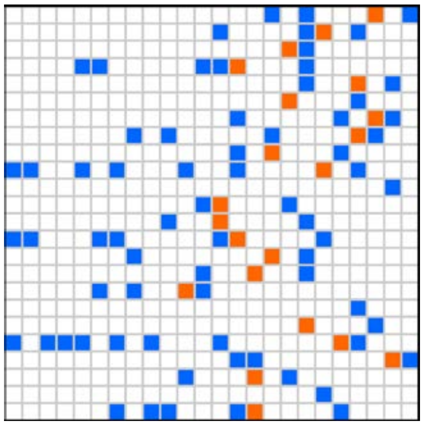
- Example: compute 10 left-most Eigenpairs for a “spin chain” of length L=20 in a magnetic field
- Fixed 10 iterations of GMRES for correction equation

Block size	# JD iters	# matvecs
1	63	693
2	37	407
4	29	319
5	23	253

Sparse MVM on heterogenous nodes

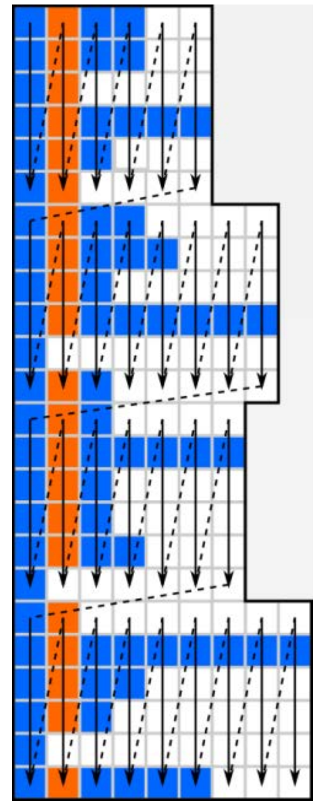


Solution: SELL $C - \sigma$ storage format

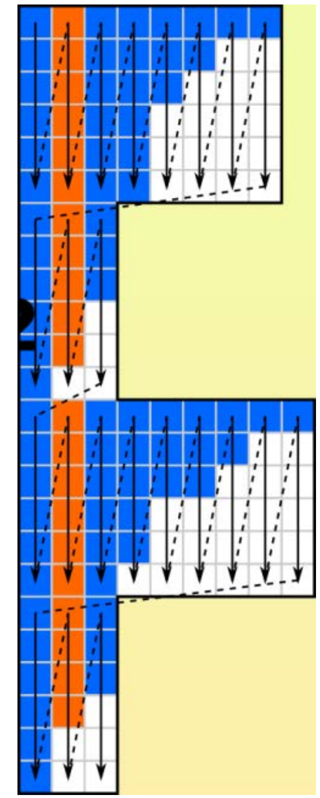


-(C)hunk size machine dependent

-(σ)orting width, matrix dependent



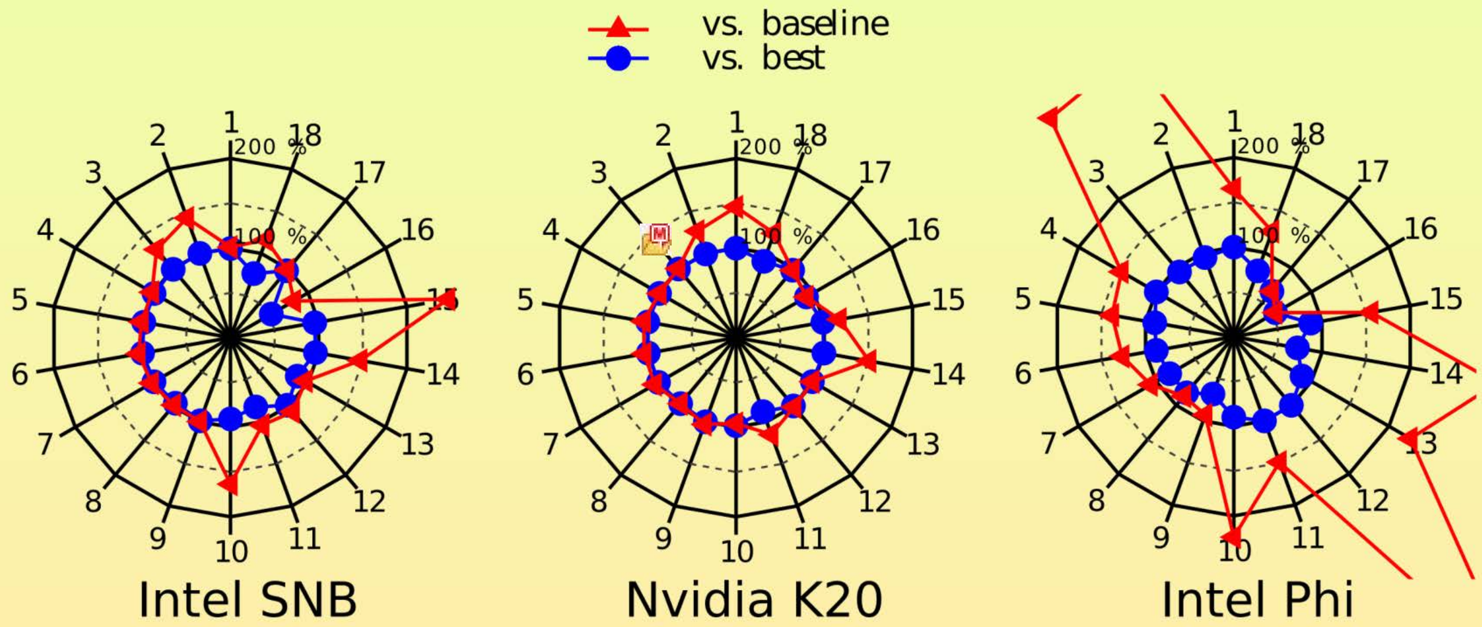
C=6, σ=1



C=6, σ=12

SELL $C-\sigma$ with fixed parameter C

Performance of unified format



How can performnace engineering help?

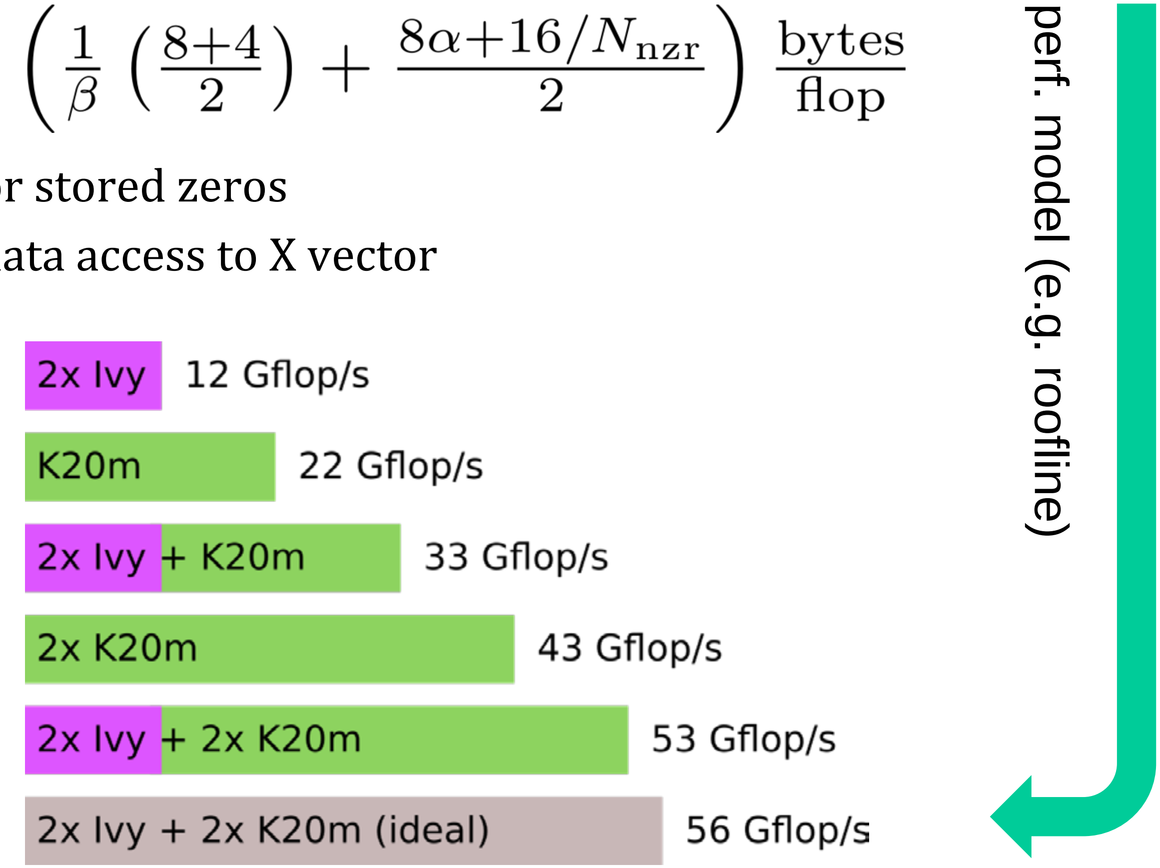
$$B_{\mathrm{SELL}}^{\mathrm{DP}} = \left(\frac{1}{\beta} \left(\frac{8+4}{2} \right) + \frac{8\alpha + 16/N_{\mathrm{nzr}}}{2} \right) \frac{\text{bytes}}{\text{flop}}$$

α: overhead for stored zeros

β: quantifies data access to X vector

M. Kreutzer, et al. *A unified sparse matrix data format for modern processors with wide SIMD units. Submitted*.

Preprint: [arXiv:1307.6209]



Current state of JD in ESSEX

- GHOST: General Hybrid Optimized Sparse Toolkit
  - efficient sparse matrices and block vectors
  - queuing system for out-of-order execution
  - written in C/C++, OpenMP, OpenCL, and CUDA
  - single/double precision, real, complex

- PHIST: Pipelined Hybrid Iterative Solver Toolkit
  - single-vector JDQR and block JD, not fully optimized yet
  - choice of numerical libraries to provide “core operations”:
    - Epetra/Tpetra (Trilinos)
    - GHOST (ESSEX)
  - Several hundred unit tests to ensure software quality
  - Callable from C/C++, Fortran, Python…

Next steps

- More optimizations possible
  - overlapping of communication and computation

- Adaptive “inner tolerance” (inexact Newton) for Block JD

- Extend performance engineering to entire algorithm

- Preconditioning for inner iteration

- Assess numerical and computational performance

contact: Jonas.Thies@dlr.de