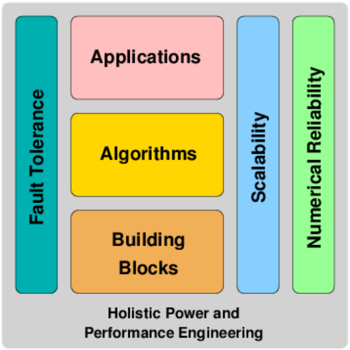
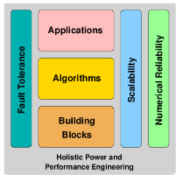
Equipping Sparse Solvers for Exascale (ESSEX / ESSEX II)

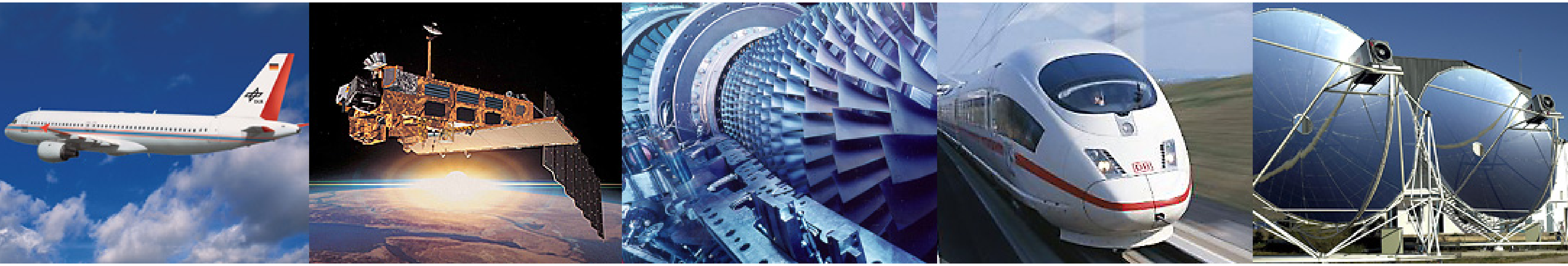


Gerhard Wellein	Computer Science, University Erlangen
Bruno Lang	Applied Computer Science, University Wuppertal
Achim Basermann	**Simulation & SW Technology, German Aerospace Center**
Holger Fehske	Institute for Physics, University Greifswald
Georg Hager	Erlangen Regional Computing Center
Tetsuya Sakurai	Applied Mathematics, University of Tsukuba
Kengo Nakajima	Computer Science, University of Tokyo

ESSEX: 2013 – 2015
ESSEX II: 2016 – 2018

DLR
German Aerospace Center





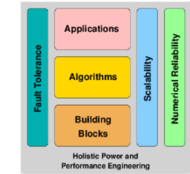
- Research Institution
- Space Agency
- Project Management Agency

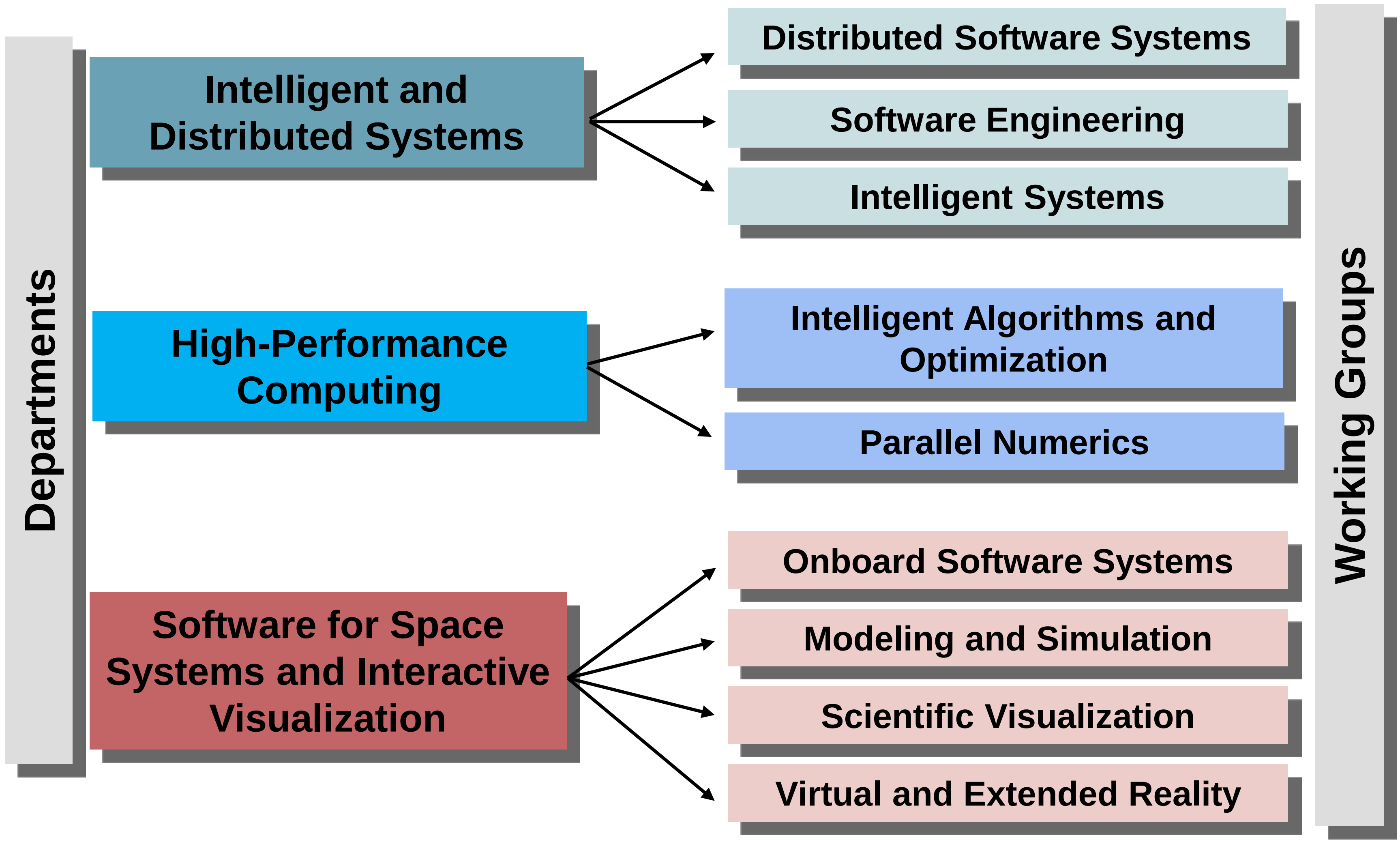
DLR Locations and Employees

Approx. 8000 employees across
40 institutes and facilities at 20 sites.

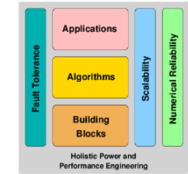
Offices in Brussels, Paris,
Tokyo and Washington

DLR Institute Simulation and Software Technology
Scientific Themes and Working Groups





High Performance Computing Teams

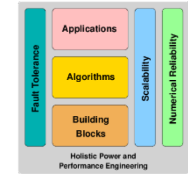


Department
High Performance Computing
Head: Dr. Achim Basermann
Deputy: Dr. Margrit Klitz

Intelligent Algorithms & Optimiziation
Dr. Martin Siggel

Quantum Computing

Parallel Numerics
Dr. Jonas Thies



- Motivation

- Software:
  - Interoperability, portability & performance
  - Supporting libraries

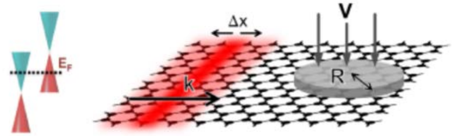
- Multicoloring and ILU Preconditioning

- Extreme Eigenvalues Computation: Jacobi-Davidson Method

- Inner Eigenvalue Computation: Filter Diagonalization

ESSEX project – background

Quantum physics/information applications



Large, Sparse

$$i\hbar\frac{\partial}{\partial t}\psi(\vec{r},t)=H\psi(\vec{r},t)$$

and beyond....

$$\boldsymbol{H}\,\boldsymbol{x} = \boldsymbol{\lambda}\,\boldsymbol{x}$$

"Few" (1,...,100s) of eigenpairs

"**Bulk**" (100s,...,1000s) eigenpairs

$$\underbrace{\{\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \ldots, \ldots, \ldots, \ldots, \boldsymbol{\lambda_k}, \ldots, \ldots, \ldots, \ldots, \boldsymbol{\lambda_{n-1}}, \boldsymbol{\lambda_n}\}}$$

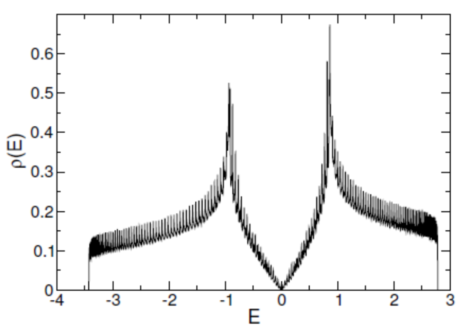
Good approximation to full spectrum (e.g. Density of States)

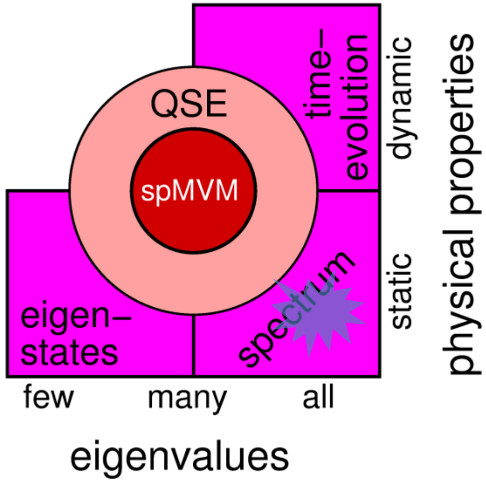
→ Sparse eigenvalue solvers of broad applicability

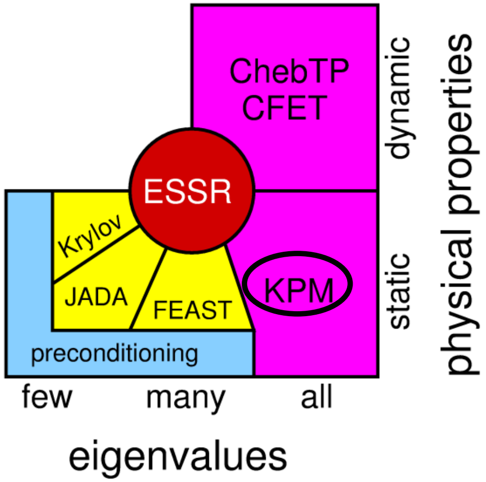
Application, Algorithm and Performance: Kernel Polynomial Method (KPM) – A Holistic View

- Compute **approximation to the complete eigenvalue spectrum** of large sparse matrix A (with $X = I$)

$$X(\omega) = \frac{1}{N}\mathrm{tr}[\delta(\omega - H)X] = \frac{1}{N}\sum_{n=1}^{N} \delta(\omega - E_n)\langle\psi_n, X\psi_n\rangle$$







The Kernel Polynomial Method (KPM)

Optimal performance exploit knowledge from all software layers!

Basic algorithm – Compute Cheyshev polynomials/moments:

Application:
Loop over random initial states

Building blocks:
(Sparse) linear algebra library

for $r = 0$ to $R-1$ **do**
    $|v\rangle \leftarrow |\mathrm{rand}()\rangle$
    Initialization steps and computation of η_0, η_1
    for $m = 1$ to $M/2$ **do**

Algorithm:
Loop over moments

        $\mathrm{swap}(|w\rangle, |v\rangle)$
        $|u\rangle \leftarrow H|v\rangle$ ▷ spmv() Sparse matrix vector multiply
        $|u\rangle \leftarrow |u\rangle - b|v\rangle$ ▷ axpy() Scaled vector addition
        $|w\rangle \leftarrow -|w\rangle$ ▷ scal() Vector scale
        $|w\rangle \leftarrow |w\rangle + 2a|u\rangle$ ▷ axpy() Scaled vector addition
        $\eta_{2m} \leftarrow \langle v|v\rangle$ ▷ nrm2() Vector norm
        $\eta_{2m+1} \leftarrow \langle w|v\rangle$ ▷ dot() Dot Product
    end for
end for

The Kernel Polynomial Method (KPM)

Optimal performance exploit knowledge from all software layers!

Basic algorithm – Compute Cheyshev polynomials/moments:

for $r = 0$ to $R-1$ **do**
- $|v\rangle \leftarrow |\mathrm{rand}()\rangle$
- Initialization steps and computation of η_0, η_1
- **for** $m = 1$ to $M/2$ **do**
  - $\mathrm{swap}(|w\rangle, |v\rangle)$
  - $|u\rangle \leftarrow H|v\rangle$ ▷ `spmv()`
  - $|u\rangle \leftarrow |u\rangle - b|v\rangle$ ▷ `axpy()`
  - $|w\rangle \leftarrow -|w\rangle$ ▷ `scal()`
  - $|w\rangle \leftarrow |w\rangle + 2a|u\rangle$ ▷ `axpy()`
  - $\eta_{2m} \leftarrow \langle v|v\rangle$ ▷ `nrm2()`
  - $\eta_{2m+1} \leftarrow \langle w|v\rangle$ ▷ `dot()`
- **end for**

end for

for $r = 0$ to $R-1$ **do**
- $|v\rangle \leftarrow |\mathrm{rand}()\rangle$
- Initialization steps and computation of η_0, η_1
- **for** $m = 1$ to $M/2$ **do**
  - $\mathrm{swap}(|w\rangle, |v\rangle)$
  - $|w\rangle = 2a(H - b\mathbb{1})|v\rangle - |w\rangle$ &
  - $\eta_{2m} = \langle v|v\rangle$ &
  - $\eta_{2m+1} = \langle w|v\rangle$ ▷ `aug_spmv()`
- **end for**

Augmented Sparse Matrix Vector Multiply

The Kernel Polynomial Method (KPM)

Optimal performance exploit knowledge from all software layers!

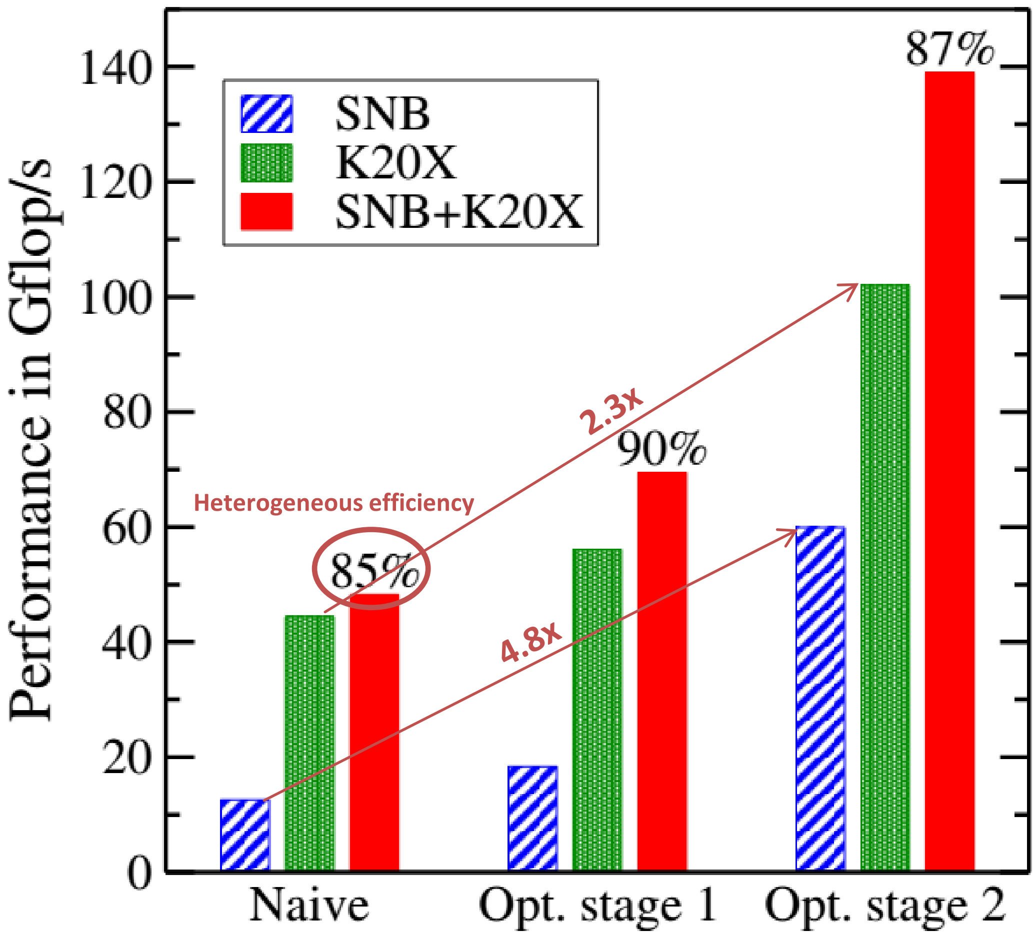
Basic algorithm – Compute Cheyshev polynomials/moments:

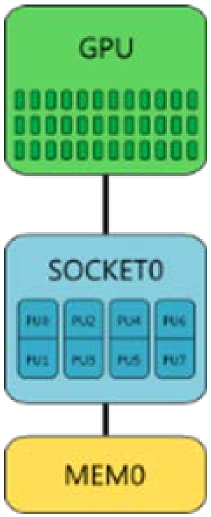
for $r = 0$ to $R-1$ **do**
  $|v\rangle \leftarrow |\text{rand}()\rangle$
  Initialization steps and computation of η_0, η_1
  for $m = 1$ to $M/2$ **do**
    $\text{swap}(|w\rangle, |v\rangle)$
    $|w\rangle = 2a(H - b\mathbb{1})|v\rangle - |w\rangle$ &
      $\eta_{2m} = \langle v|v\rangle$ &
      $\eta_{2m+1} = \langle w|v\rangle$ ▷ `aug_spmv()`
  end for

$|V\rangle := |v\rangle_{0..R-1}$ ▷ Assemble vector blocks
$|W\rangle := |w\rangle_{0..R-1}$
$|V\rangle \leftarrow |\text{rand}()\rangle$
Initialization steps and computation of μ_0, μ_1
for $m = 1$ to $M/2$ **do**
  $\text{swap}(|W\rangle, |V\rangle)$
  $|W\rangle = 2a(H - b\mathbb{1})|V\rangle - |W\rangle$ &
    $\eta_{2m}[:] = \langle V|V\rangle$ &
    $\eta_{2m+1}[:] = \langle W|V\rangle$ ▷ `aug_spmmv()`
end for

Augmented Sparse Matrix
Multiple Vector Multiply

KPM: Heterogenous Node Performance



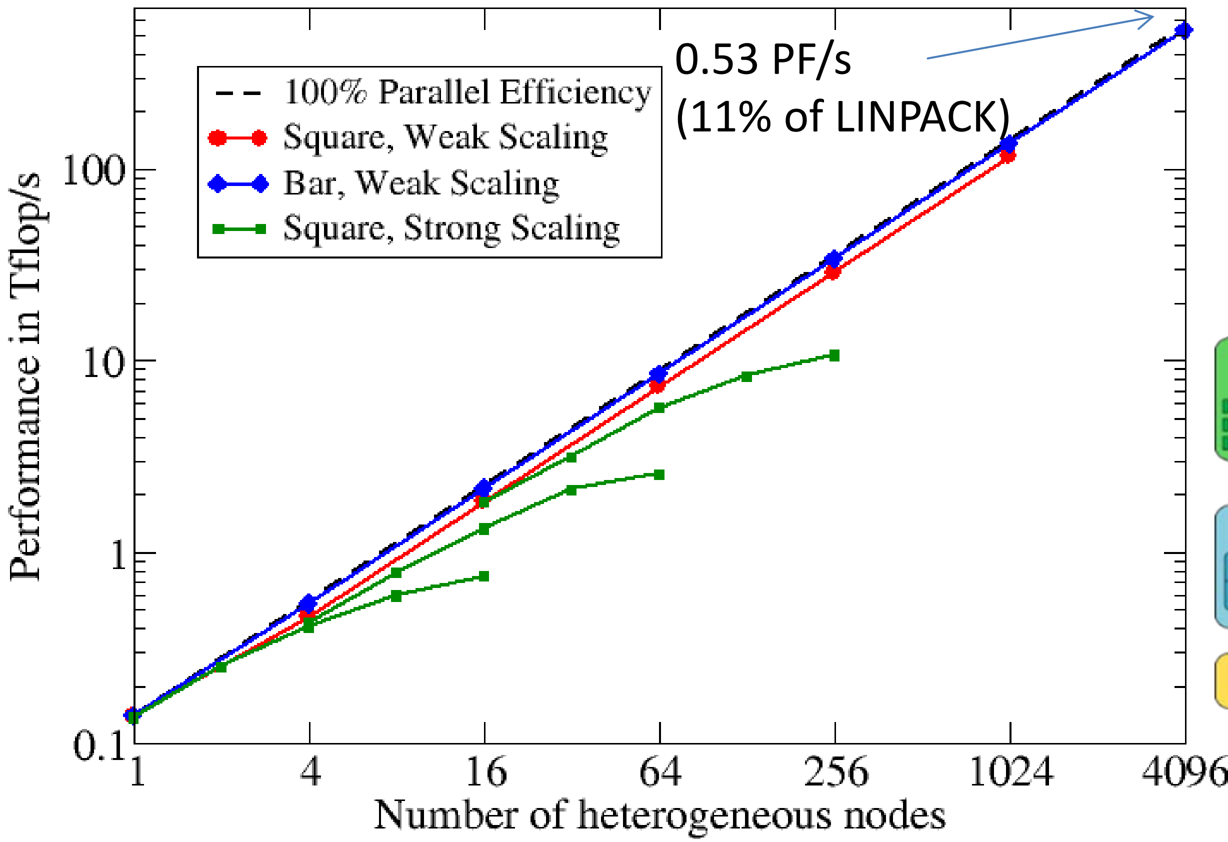


NVIDIDA K20X

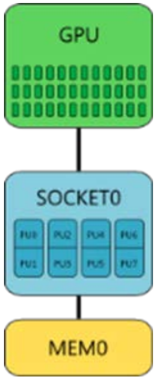
Intel
Xeon E5-2670 (SNB)

- Topological Insulator Application
- Double complex computations
- Data parallel static workload distribution

KPM: Large Scale Heterogenous Node Performance



CRAY XC30 – PizDaint*

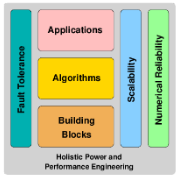


- 5272 nodes
- Peak: 7.8 PF/s
- LINPACK: 6.3 PF/s
- Largest system in Europe

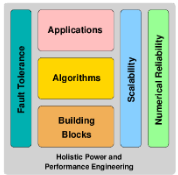
Performance Engineering of the Kernel Polynomial Method on Large-Scale CPU-GPU Systems
M. Kreutzer, A. Pieper, G. Hager, A. Alvermann, G. Wellein and H. Fehske, IEEE IPDPS 2015

*Thanks to CSCS/T. Schulthess for granting access and compute time

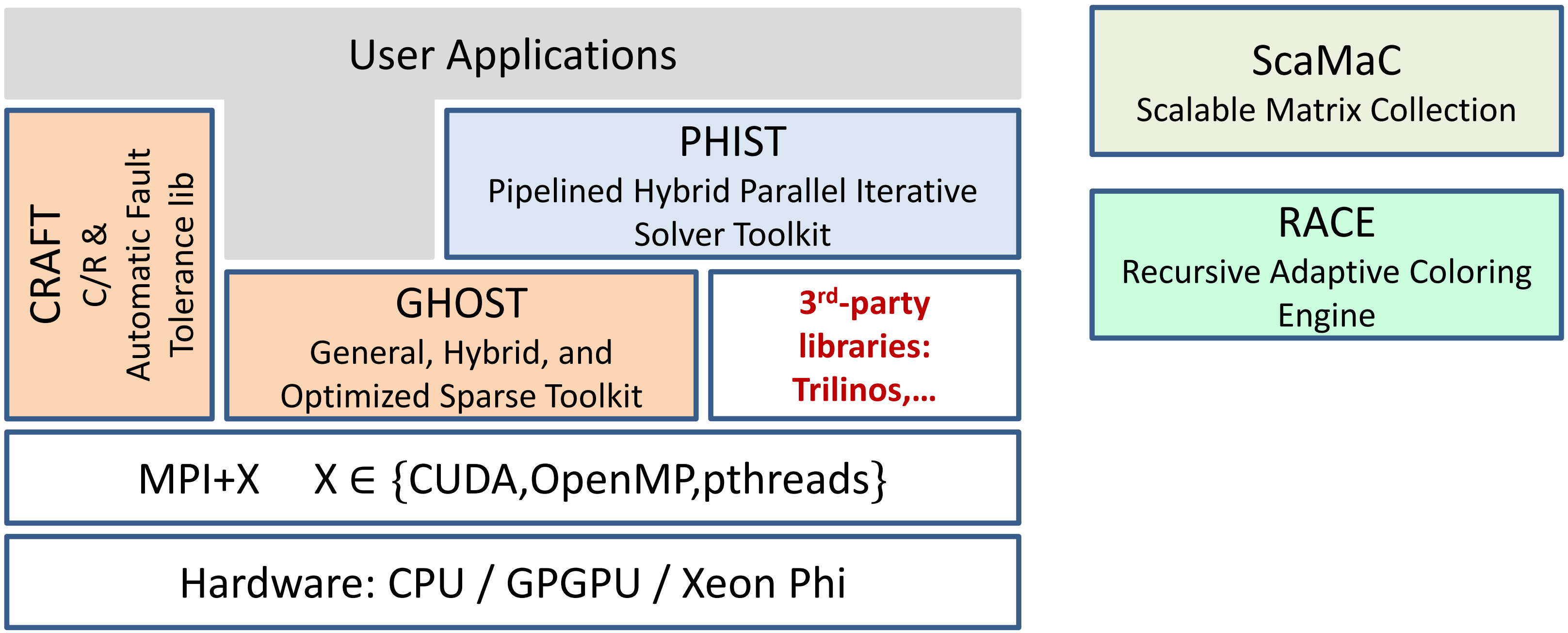
Motivated by quantum physics applications



ESSEX - I		ESSEX - II
Conservative Quantum Systems	→	Dissipative Quantum Systems
$A^{+} = A$	→	$A^{+} \neq A$
$Ax = \lambda x$	→	$Ax = \lambda Bx$ $A(\lambda)x = 0$
ESSR: Blueprints	→	Interoperable Library



ESSEX-II: Software Packages



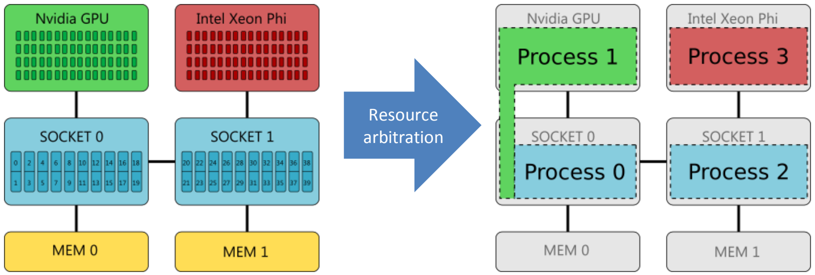
Links to open source repositories at **https://blogs.fau.de/essex/code**

Software: Interoperability portability & performance

Kernel library (GHOST) and solver framework (PHIST)

GHOST library

- **Hybrid MPI+X** execution mode
  (X=OpenMP, CUDA)



- **Algorithm specific kernels**: SIMD Intrinsics (KNL) and CUDA (NVIDIA)
  → 2x – 5x speed-up vs. Optimized general building block libraries

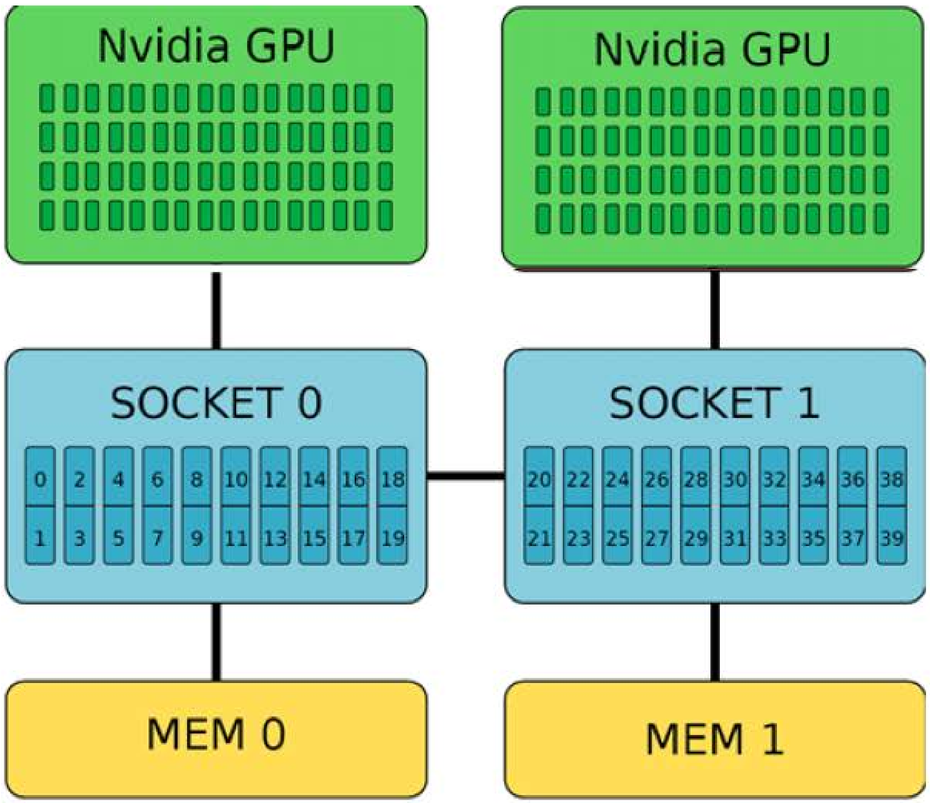
- **Tall & skinny matrix-matrix kernels** (block orthogonalization)
  → 2x – 10x speed-up vs. Optimized general building block libraries

- **SELL-C-σ sparse matrix format**

- Open Source code & example applications: https://bitbucket.org/essex/ghost

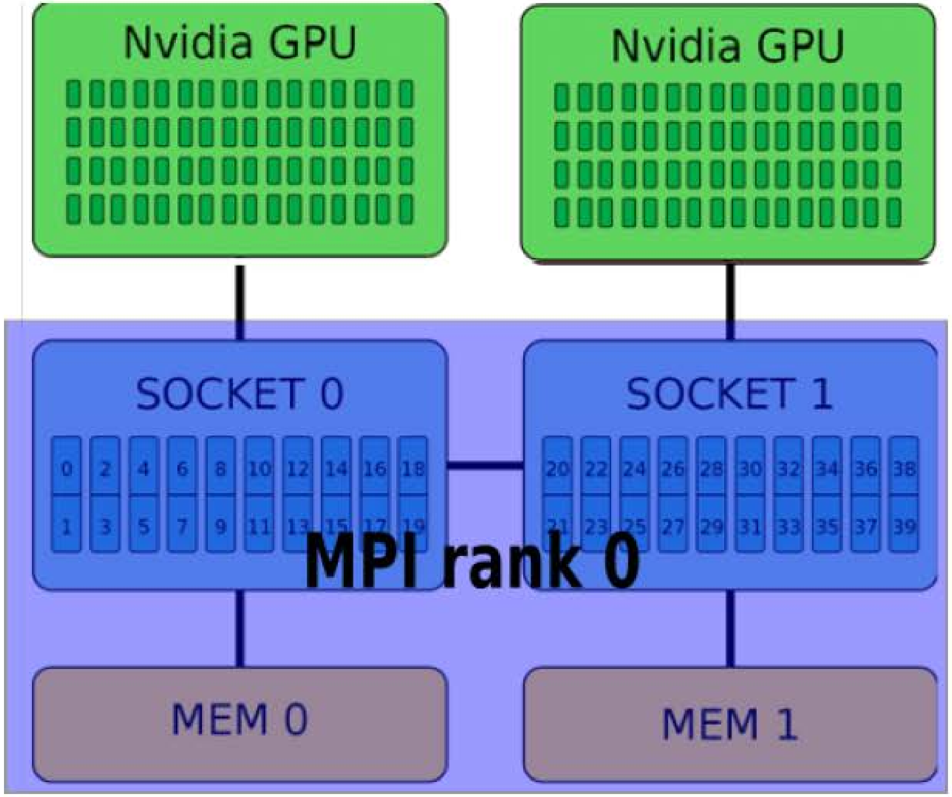
The ESSEX Software Infrastructure: MPI + X with GHOST

- System with multiple CPUs (NUMA domains) and GPUs



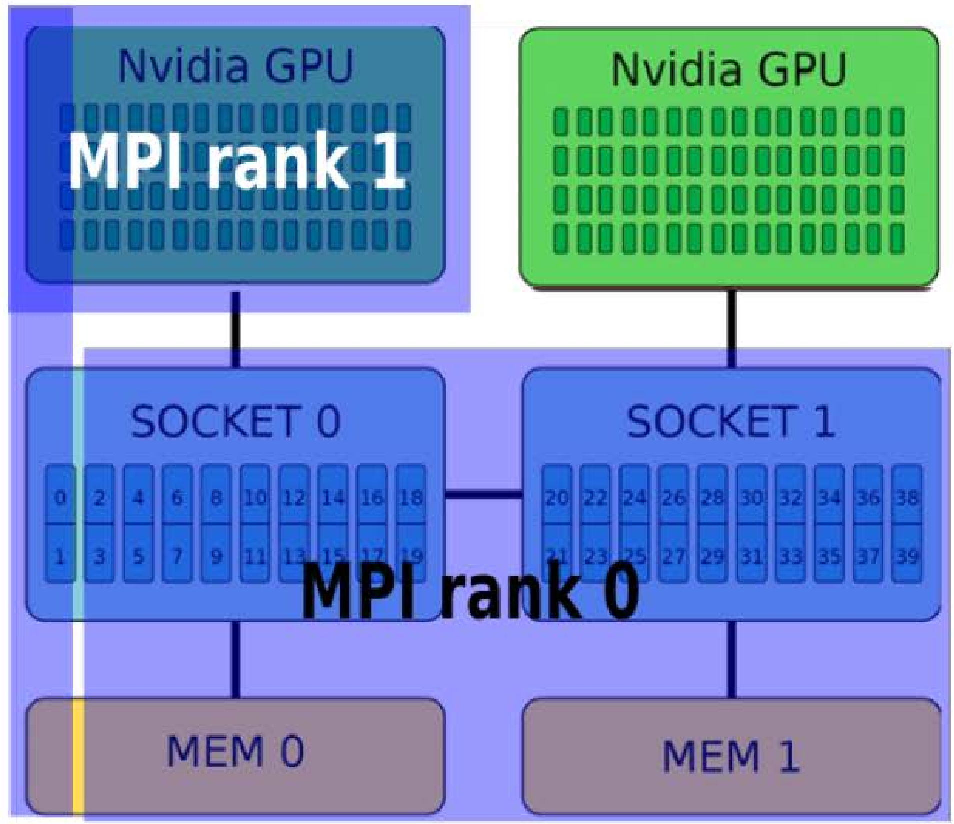
The ESSEX Software Infrastructure: MPI + X with GHOST

- System with multiple CPUs (NUMA domains) and GPUs
- -np 1: use entire CPU



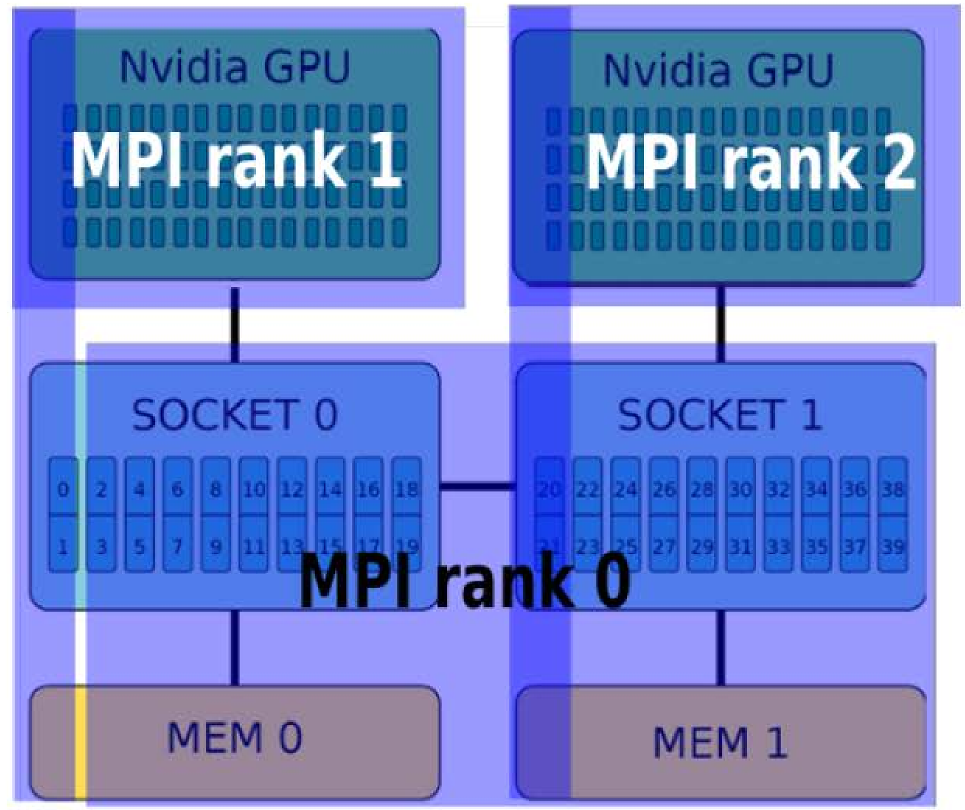
The ESSEX Software Infrastructure: MPI + X with GHOST

- System with multiple CPUs (NUMA domains) and GPUs
- -np 1: use entire CPU
- -np 2: use CPU and first GPU



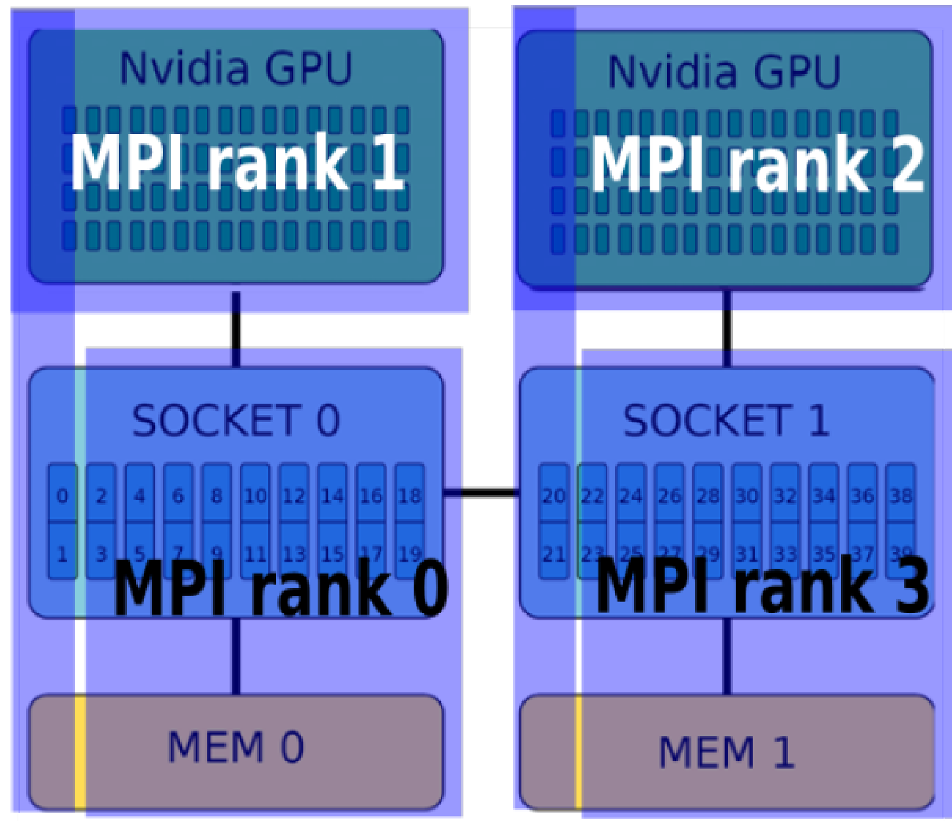
The ESSEX Software Infrastructure: MPI + X with GHOST

- System with multiple CPUs (NUMA domains) and GPUs
- -np 1: use entire CPU
- -np 2: use CPU and first GPU
- -np 3: use CPU and both GPUs



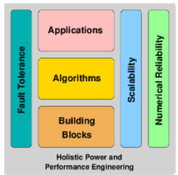
The ESSEX Software Infrastructure: MPI + X with GHOST

- System with multiple CPUs (NUMA domains) and GPUs
- -np 1: use entire CPU
- -np 2: use CPU and first GPU
- -np 3: use CPU and both GPUs
- -np 4: use one process per socket and one for each GPU



Option: distribute problem according to memory bandwidth measured

A Portable and Interoperable Eigensolver Library



PHIST (Pipelined Hybrid Parallel Iterative Solver Toolkit) sparse solver framework

- General-purpose block Jacobi-Davidson Eigensolver, Krylov methods
- Preconditioning interface
- C, C++, Fortran 2003 and Python bindings
- Backends (kernel libs) include **GHOST**, Tpetra, PETSc, Eigen, Fortran
- Can use Trilinos solvers Belos and Anasazi, independent of backend

Atlassian Bitbucket

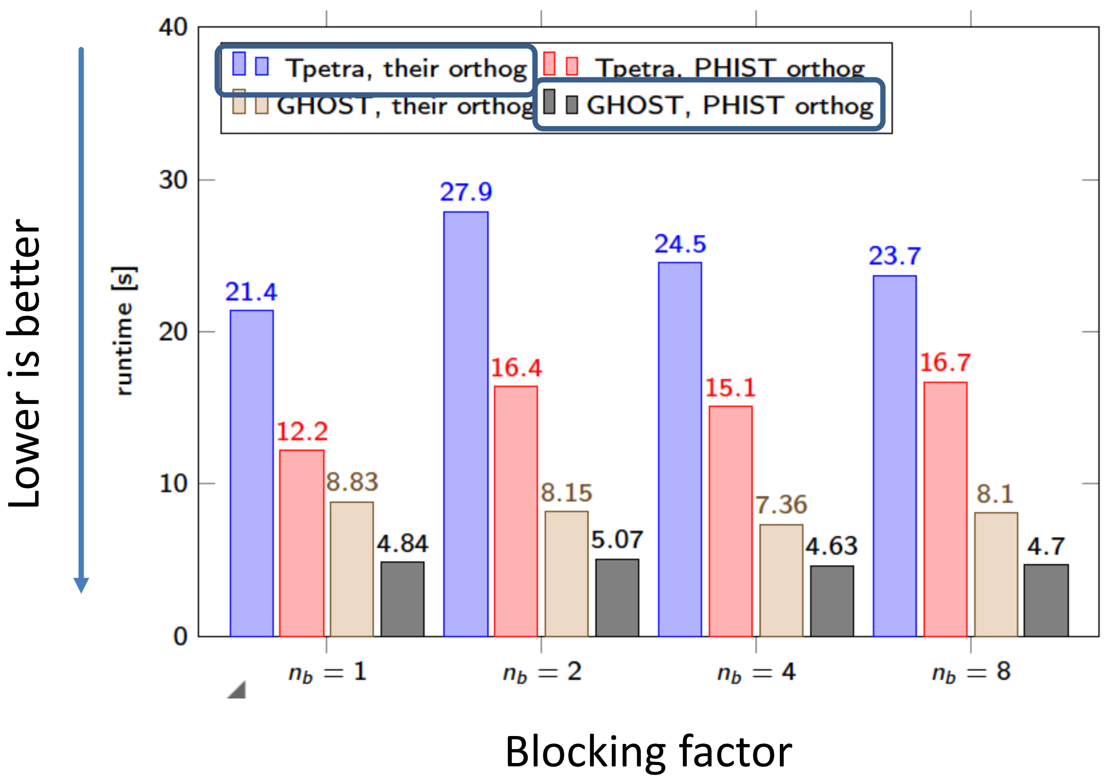
Getting PHIST and GHOST

- https://bitbucket.org/essex/[ghost,phist]
- Cmake build system
- Availale via Spack
- https://github.com/spack/spack/
- PHIST will join **Extreme-Scale Development Kit**, https://xSDK.info/

PHIST & GHOST – interoperability & performance

- **Anasazi** Block Krylov-Schur **solver** on **Intel Skylake CPU**

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



- **Anasazi's** kernel interface mostly a subset of PHIST → extends PHIST by e.g. BKS and LOBPCG

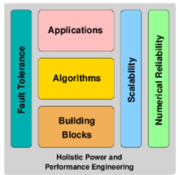
- Trilinos not optimized for block vectors in **row-major storage**

Anasazi: https://trilinos.org/packages/anasazi/
Tpetra: https://trilinos.org/packages/tpetra/

Software: Supporting libraries

FT/CR library (CRAFT) and matrix generation (ScaMac)

CRAFT library: Application-level Checkpoint/Restart & Automatic Fault Tolerance



Application-level Checkpoint/Restart(CR):

- Simple & extendable interface to integrate CR functionality with minimal code changes
- Node-level CR using SCR, asyn. CP., Multi-stage & Nested CPs, signal based CP

Automatic Fault Tolerance (AFT) using CR

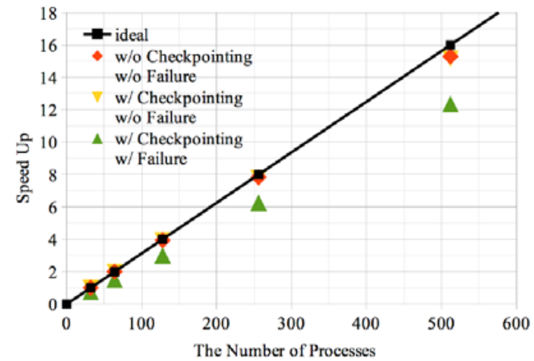
- Define \`AFT-zones´ for automatic communication recovery in case of process failures.
- Detection and recovery methods from User-level Failure Mitigation (ULFM) **MPI-ULFM**.

Goal: Low programming & performance overhead

Tested Applications:

- GHOST & PHIST applications from ESSEX
- pFEM-CRAFT [Nakajima (U.Tokyo)] →

https://bitbucket.org/essex/craft

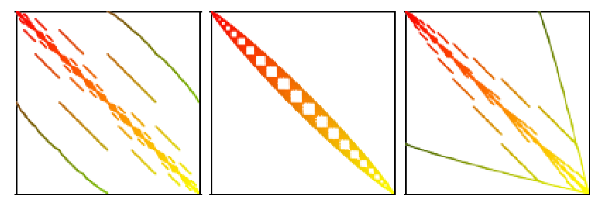


ScaMaC: Scalable Matrix Collection

Goal: Collection of parametrized sparse matrices for eigenvalue computations from (quantum) physics

Features:

- „Scalable" matrix generator instead of fixed-size matrices
- Compatible with PETSc, Trilinos, GHOST, PHIST ...
- „Real World" (quantum) physics matrices, e.g.
  - wave & advection-diffusion eqs.,
  - correlated systems,
  - graphene & topological insulators,
  - quantum optics, (c)QED, optomechanics,...

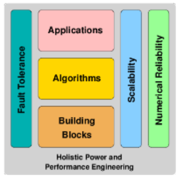


- Real & complex, symmetric & non-symmetric, easy & hard to solve matrices
- Generating matrices of dimension 10^{11} in less than 30s on full scale OFP (0,5 Mcores)

Multicoloring and ILU Preconditoning

RACE and ILU preconditioning

Recursive algebraic coloring engine (RACE)

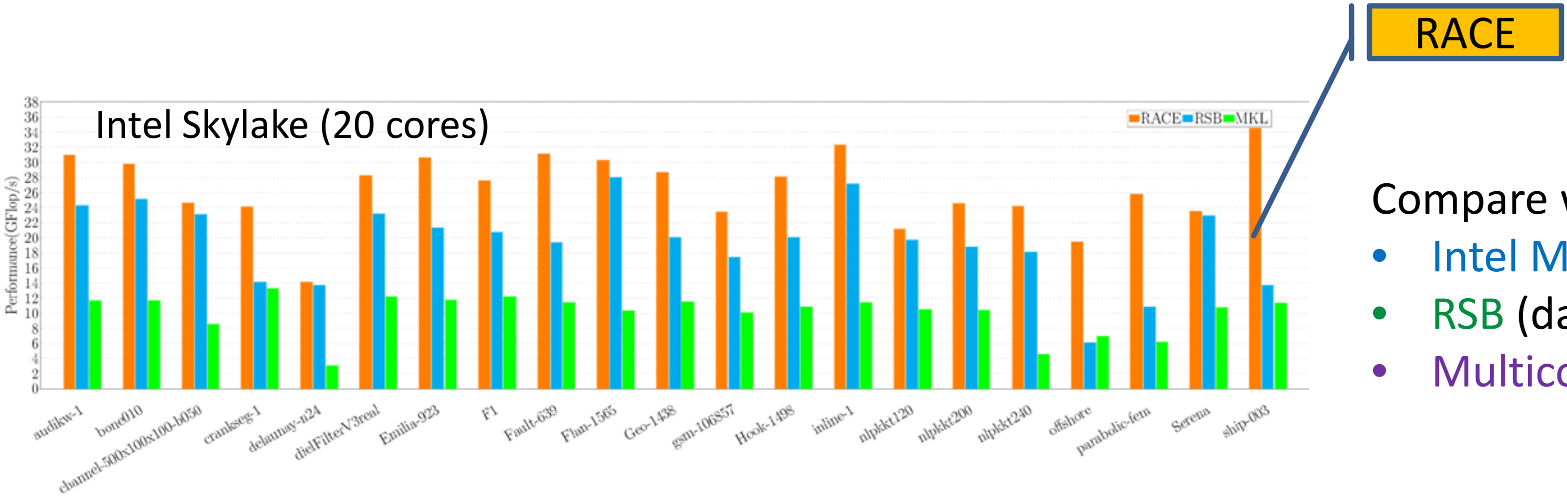


Graph coloring: RACE uses recursive BFS level based method for “distance-k coloring” of symmetric matrices

Objectives
- Preserve data locality
- Generate sufficient parallelism
- Reduce synchronization
- Simple data format like CRS

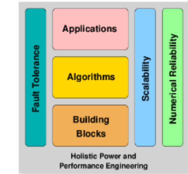
Applications – Parallelization of
- iterative solvers, e.g. Gauß-Seidel & Kaczmarz
- sparse kernels with dependencies, e.g. symmetric spMVM

Example: Node-level parallelization of symmetric spMVM (distance-2)



Compare with
- Intel MKL
- RSB (data format)
- Multicoloring

Recursive algebraic coloring engine (RACE)



Graph coloring: RACE uses recursive BFS level based method for "distance-k coloring" of symmetric matrices

Objectives

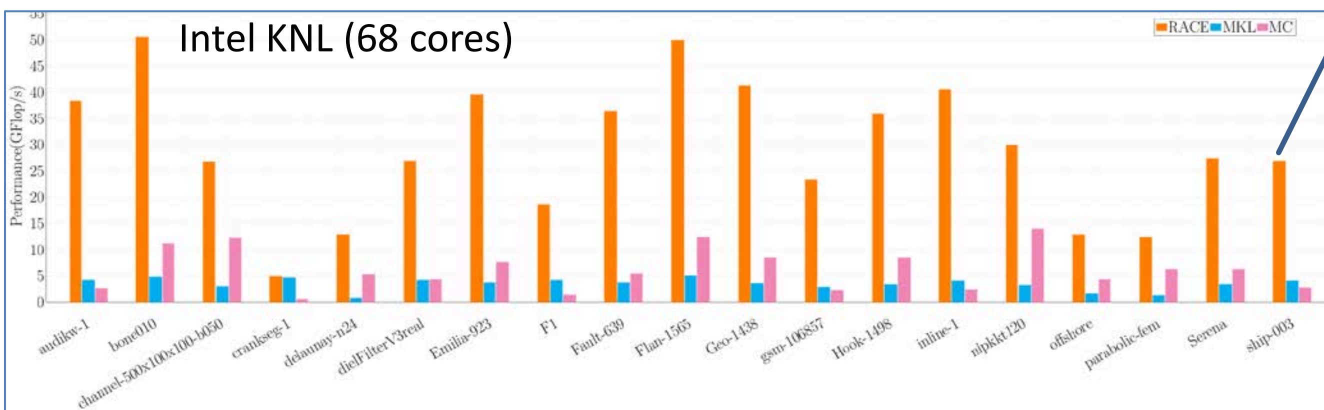
- Preserve data locality
- Generate sufficient parallelism
- Reduce synchronization
- Simple data format like CRS

Applications – Parallelization of

- iterative solvers, e.g. Gauß-Seidel & Kaczmarz
- sparse kernels with dependencies, e.g. symmetric spMVM

Example: Node-level parallelization of symmetric spMVM (distance-2)

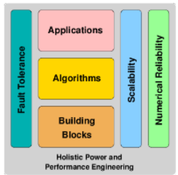
RACE



Compare with

- Intel MKL
- RSB (data format)
- Multicoloring

Integration of pKopen-SOL ILU in PHIST



- Eigensolvers in ESSEX-II (BEAST/BJDQR) require strong preconditioners for solving ill-conditioned linear systems
- PHIST has a Fortran'03 interface and backend…

with some modifications we could fully integrate a complete pKopen algorithm!

Work on pKopen-SOL ILU:

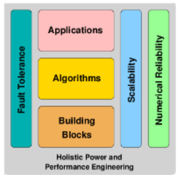
- applied 2 regularizations for robustness and better convergence (blocking and diagonal shifting)
- efficient hierarchical multi-coloring for extreme scaling

Work on PHIST:

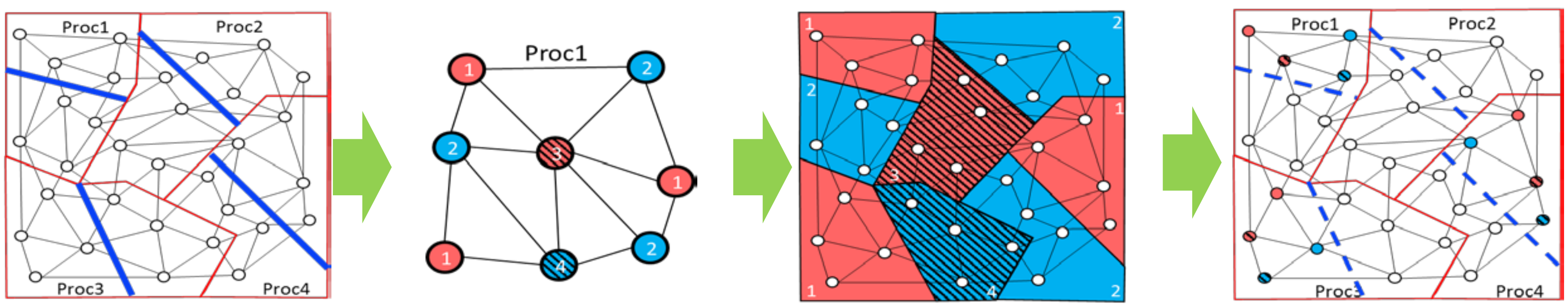
- extend matrix format to support block CRS
- Implement node-level performance models

Professional software workflow using git branches/pull requests and feature tests

Robustness & Scalability of ILU preconditioning



- Hierarchical parallelization of multi-colorings for ILU precond.



- High precision Block ILU preconditioning

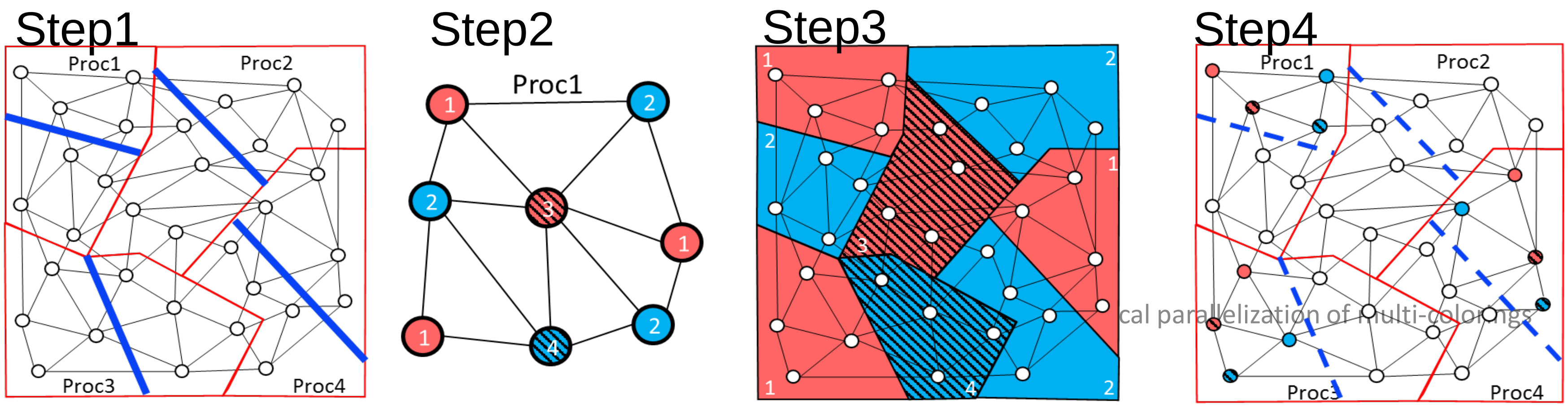
Tokyo Univ.: Masatoshi Kawai (now Riken) , Kengo Nakajima et al.

- Apply algebraic block multi-coloring to ILU preconditioning: 2.5x – 3.5x speed-up vs multicoloring

Hokkaido Univ.: Takeshi Iwashita et al.

pKopen-SOL:Parallel multi-coloring for ILU

- Proposed a hierarchical parallelization of multi-colorings



- Achieved almost constant iterations and good scalability with a graphene model (500 million DoF).
- Entire code PHIST+ILU runs on large Japanese systems Oakforest-PACS and FX10

Extreme Eigenvalues Computation: Jacobi-Davidson Method

PHIST Routine

Scalability on Oakforest-PACS
since 6 / 2018 number 12 of TOP 500

Cores: Memory: Processor: Interconnect:	556,104 919,296 GB Intel Xeon Phi 7250 68C 1.4GHz (KNL) Intel Omni-Path
Linpack Performance (Rmax) Theoretical Peak (Rpeak)	13.554 PFlop/s 24.913 PFlop/s
Nmax HPCG [TFlop/s]	9,938,880 385.479

Impression of the Oakforest-PACS supercomputer at the Japanese joint center for advanced HPC (JCAHPC).

Extreme eigenvalue computation with block Jacobi-Davidson

Goal:

Find eigenpairs (λ_j, v_j) of a large sparse matrix in a certain target space of the spectrum:

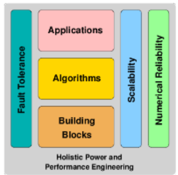
$$\boldsymbol{A} v_j = \lambda_j v_j$$

- Project the problem to a suitable subspace
- Solve the resulting small eigenproblem
- Solve the correction equation
- Orthogonalize to all previous search directions
- Extend the subspace

→ Block variant: Compute the correction equation for n_b EV concurrently
→ Limit global synchronization by exploitation of block vectors
→ Concurrently solve linear systems of equations in separate Krylov spaces
→ Combine computation of spMMVM and inner products
→ Store all block vectors row-wise

Röhrig-Zöllner, M., Thies, J., Kreutzer, M., Alvermann, A., Pieper, A., Basermann, A., Hager, G., Wellein, G., Fehske, H. (2015). Increasing the Performance of the Jacobi--Davidson Method by Blocking. *SIAM Journal on Scientific Computing*, *37*(6), C697-C722.

Benchmarks



- Fixed number of 250 Jacobi-Davidson iterations
- No additional preconditioning

Matrices

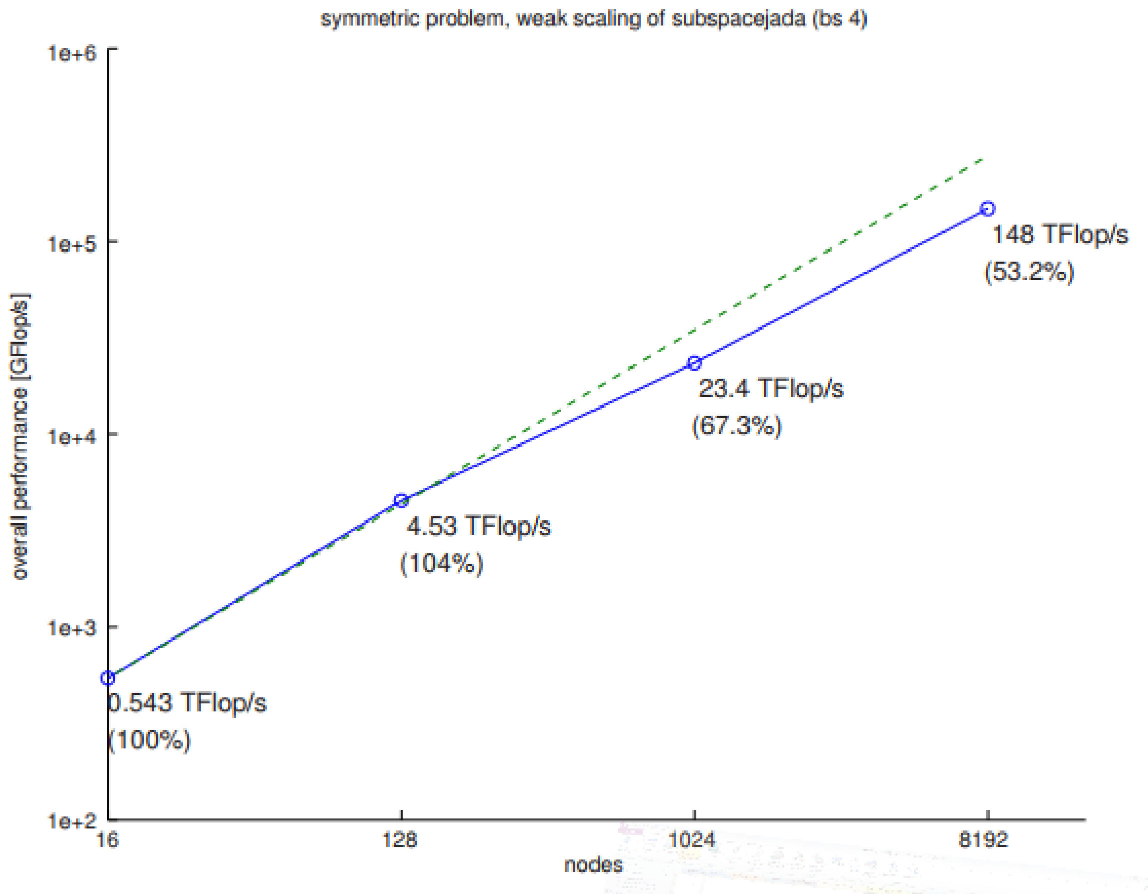
- Symmetric 7-point Laplace, 8.4M rows/node
- General 7-point, some PDE, 2.0M rows/node

Solver parameters

- Krylov solver 10 iterations of MINRES (sym.)
- or GMRES+IMGS ortho (general)
- JD basis 16-40 vectors
- target eigenpairs near 0

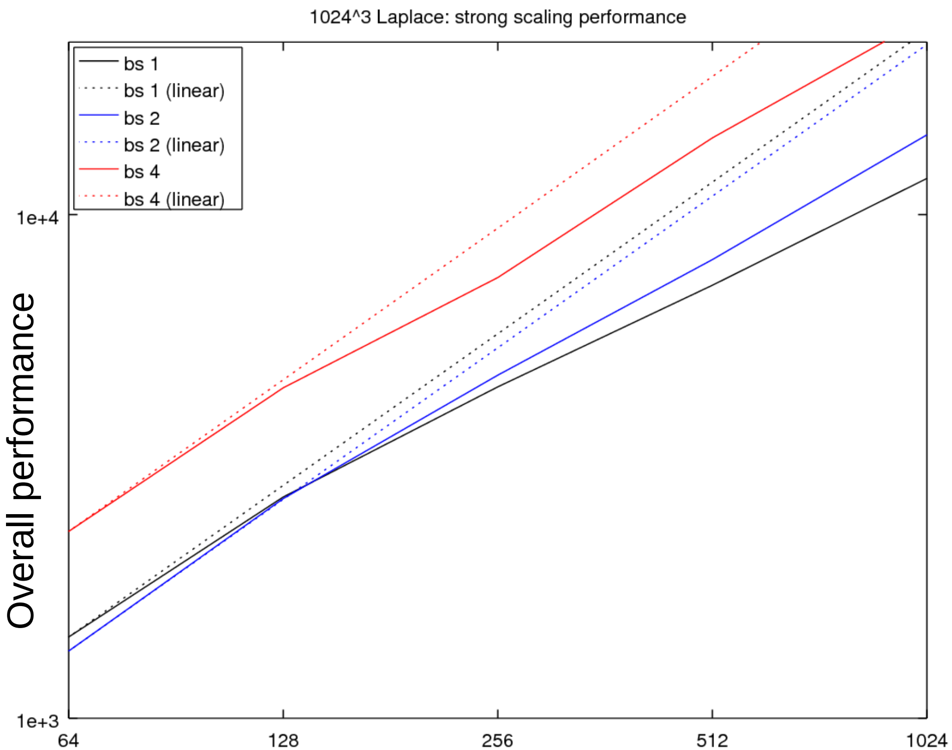
Weak scaling

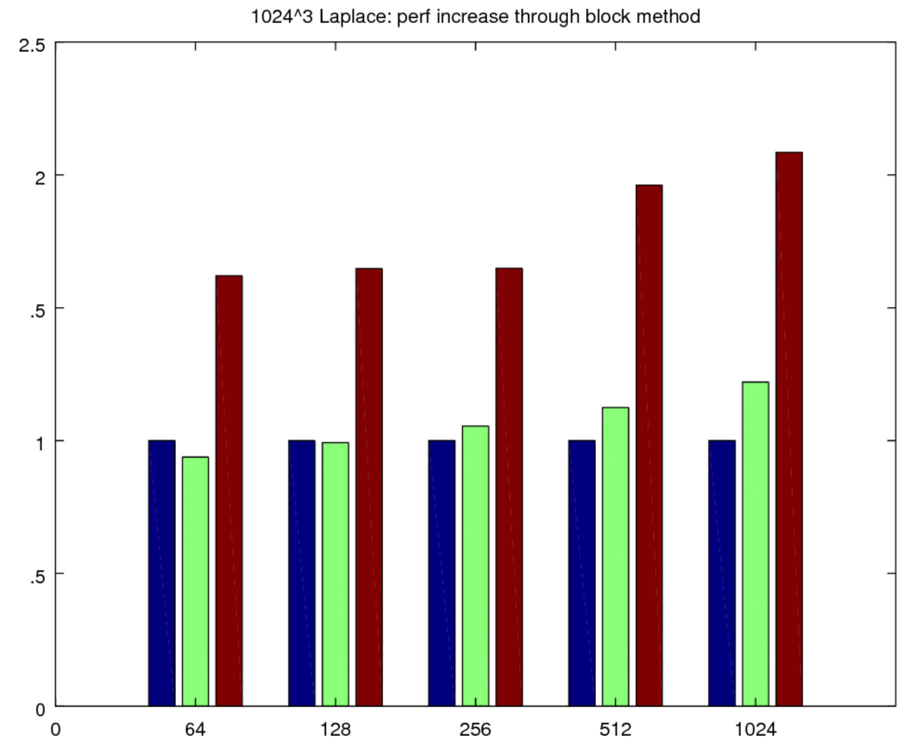
- Up to 0.5M cores
- Percentage indicates the parallel efficiency compared to the first measurement (smallest node count).
- Symmetric PDE problem with the largest matrix size N = 40 963,
- The best performance was obtained with a block size of 4.



Strong scaling

- Larger block size reduces number of Allreduce operations.





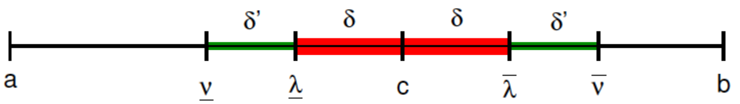
- corresponding 'block speedup' over the bs=1 case.
- The KNL doesn't seem to 'like' block size 2 very much (in contrast to XeonCPUs).
- Maybe the bandwidth can't be saturated with SSE?

Inner Eigenvalue Computation: Filter Diagonalization

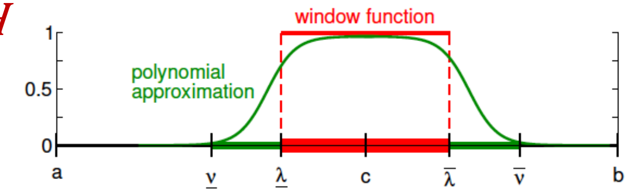
BEAST-P

Filter diagonalization - basics

- Compute all eigenpairs in $[\underline{\lambda}, \overline{\lambda}]$ within spectrum $[a, b]$ of sparse matrix H (of dimension n)



- Filter diagonalization - idea:
  - Use window function for projection onto search intervall
  - Approximate window function by polynomial in H



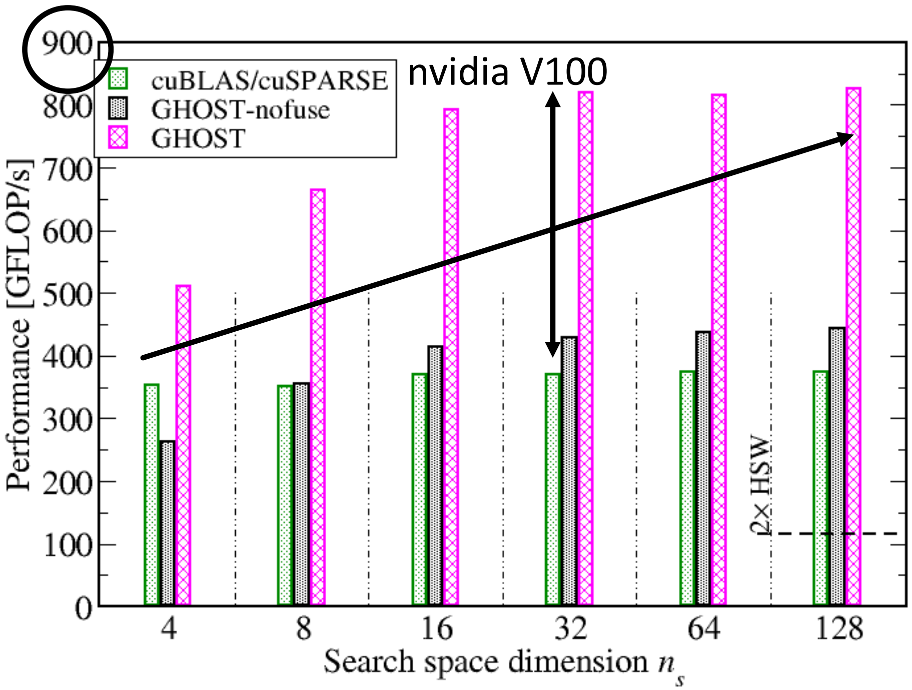
- Basic scheme:
  - Apply polynomial filter to set of search vectors ($n_S = 10^2, \ldots, 10^3$ in our case)
  - Orthogonalize filtered vectors
  - Compute Ritz-pairs and restart if neccessary

- Chebyshev Polynomials to construct filter with $H \to \widetilde{H}$ such that $\widetilde{\lambda} \in [-1, 1]$

$$T_{n+1}(\widetilde{H}) = 2\,\widetilde{H} T_n(\widetilde{H}) - T_{n-1}(\widetilde{H})$$

Performance Engineering: Optimized GHOST kernels

- Kernel: Series of BLAS1 calls and sparse matrix multiple vector multiplication (spmmv)
- GHOST: All BLAS1 calls fused with spmmv → increased intensity



Block vector width

$$n = 2.1 \times 10^6, n_p = 500$$

$$I(n_s) = \frac{146}{80 + 260/n_s} \frac{F}{B}$$

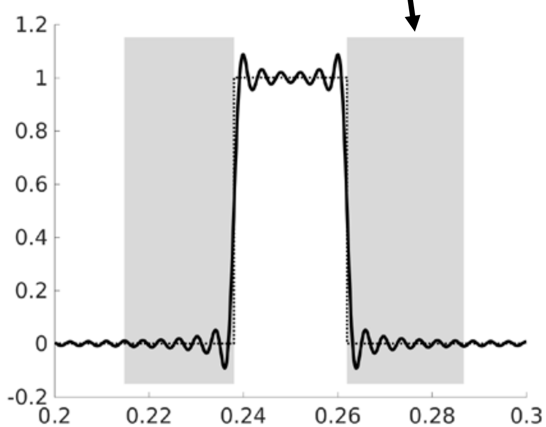
Performance increases with block vector width (row-major storage!)

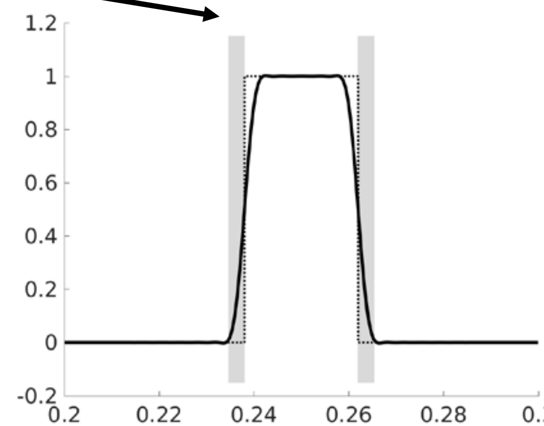
2x speed-up by kernel fusion → increases kernel complexity!

>10% of peak performance for sparse matrix problem!

Algorithm Engineering

- Improve filter quality → reduce filter degree → reduce sparse matrix vector products
- Idea: Filter must be below threshold for $[\underline{\lambda} - \delta, \overline{\lambda} + \delta]$
- Goal: Minimize δ (Lanzos smoothing kernel)



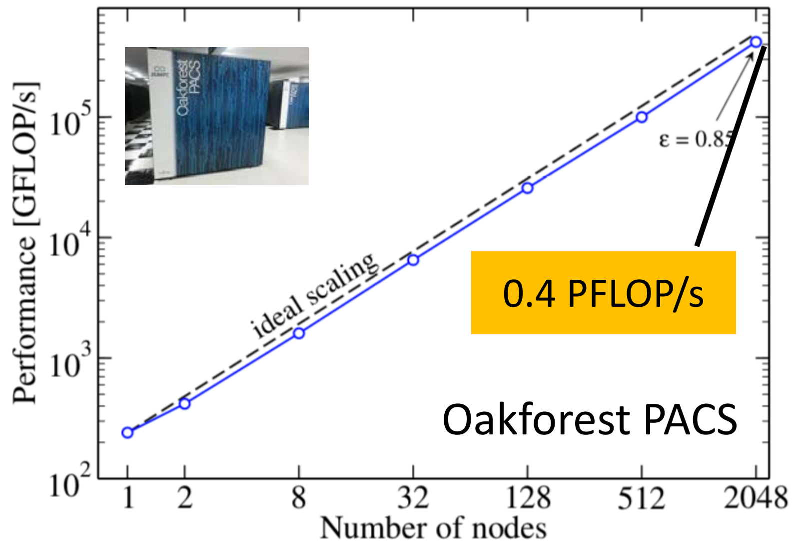


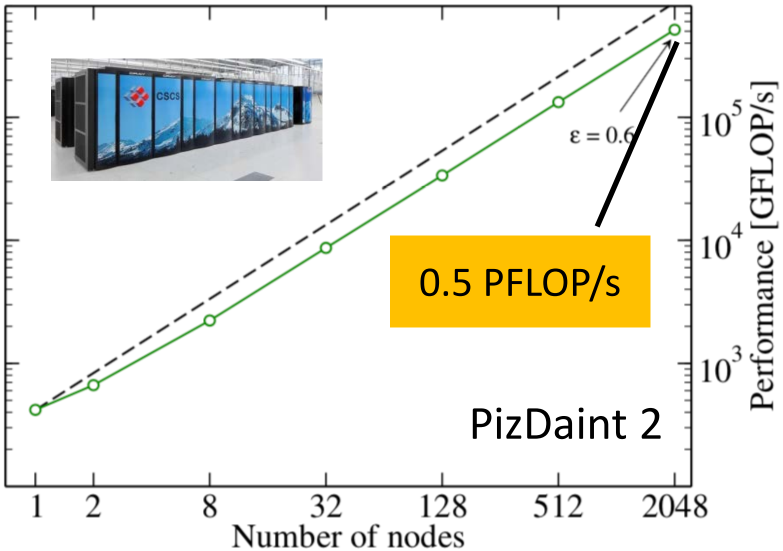
- In practice: 30-50% lower degrees → equivalent savings in time

Large scale performance – weak scaling

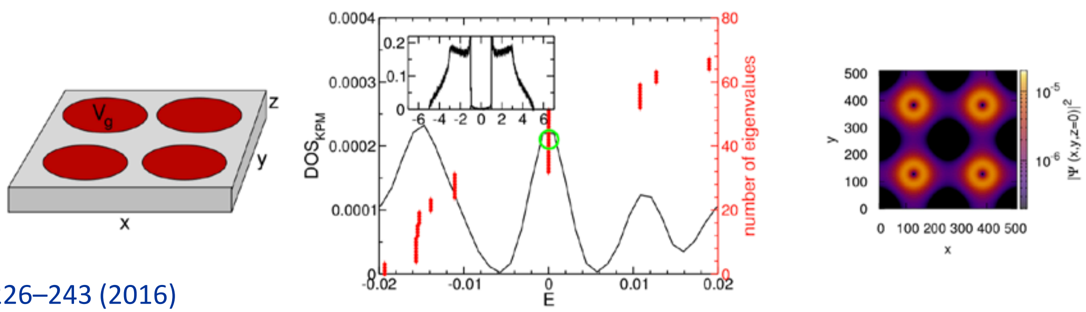
Computing 100 inner eigenvalues on matrices up to $n = 4 \times 10^9$

$$\frac{n}{node} = 2.1 \times 10^6$$
$$n_p = 500;\ n_s = 128$$



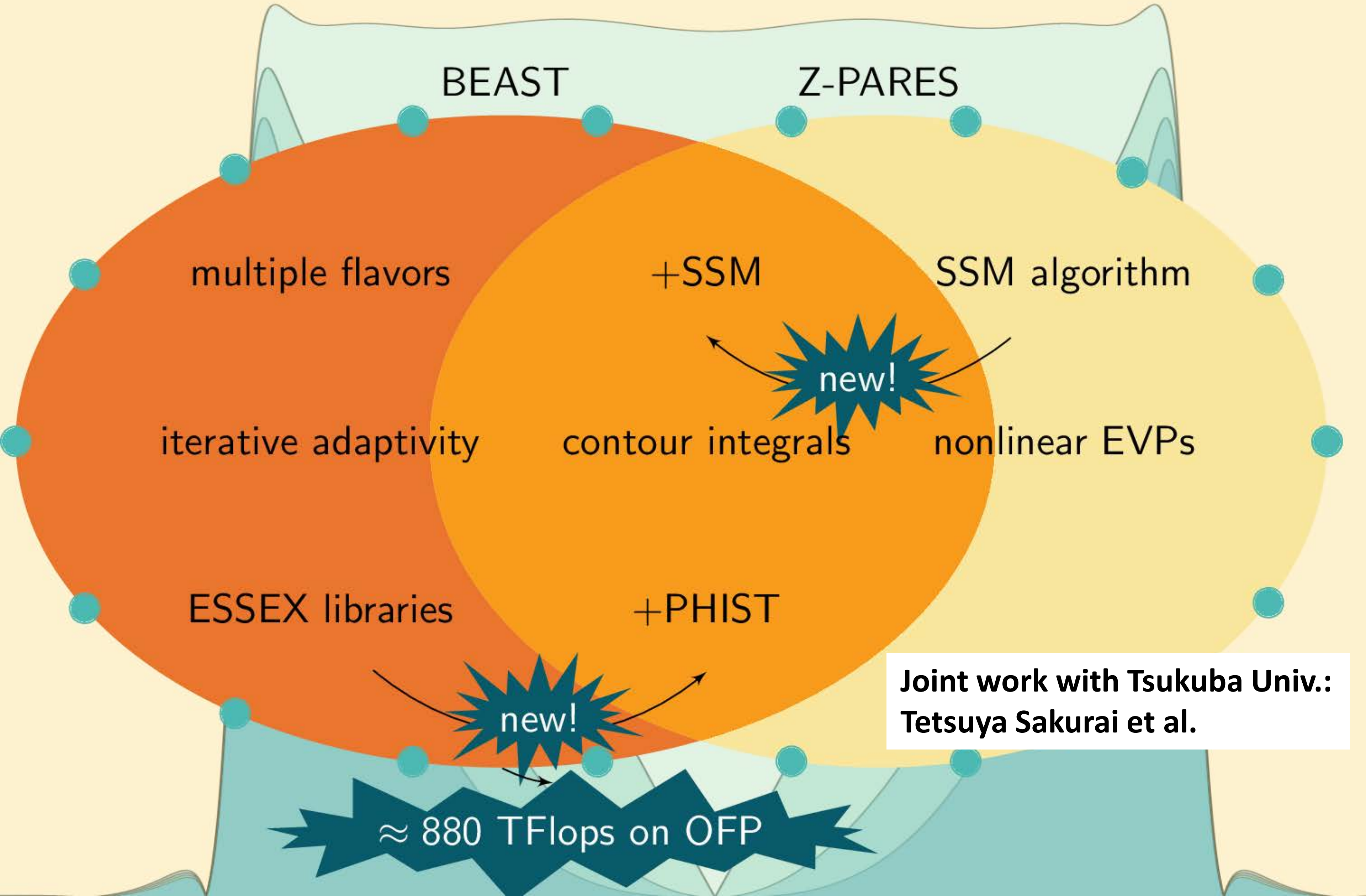


Typical Application[1]: Topological Insulator



[1] Pieper, A., et al. Journal of Computational Physics 325, 226–243 (2016)

BEASD and Z-PARES: shared tools for large EVPs



Joint work with Tsukuba Univ.:
Tetsuya Sakurai et al.

Visit our homepage: https://blogs.fau.de/essex/

THANK YOU!