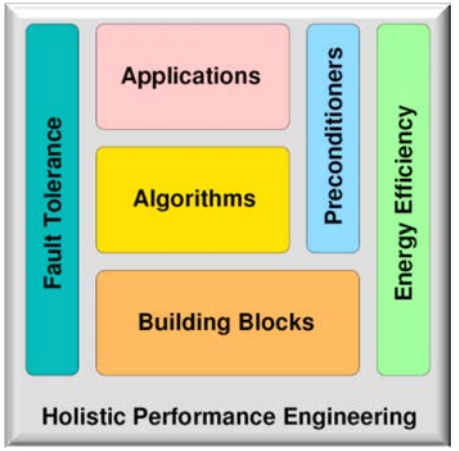
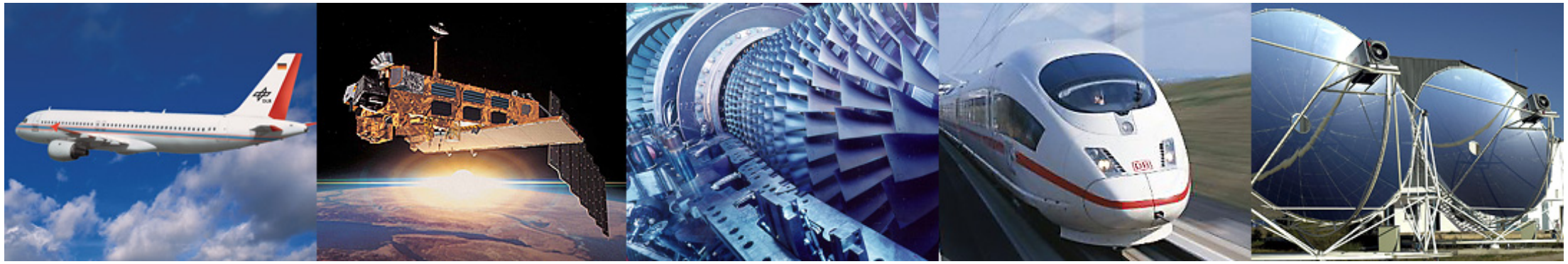
DFG Projekt ESSEX



Highly scalable sparse eigensolvers for large quantum physics problems on heterogeneous computing systems

Achim Basermann, Jonas Thies, Melven Röhrig-Zöllner
German Aerospace Center (DLR)
Simulation and Software Technology
Linder Höhe, Cologne, Germany

DLR
German Aerospace Center

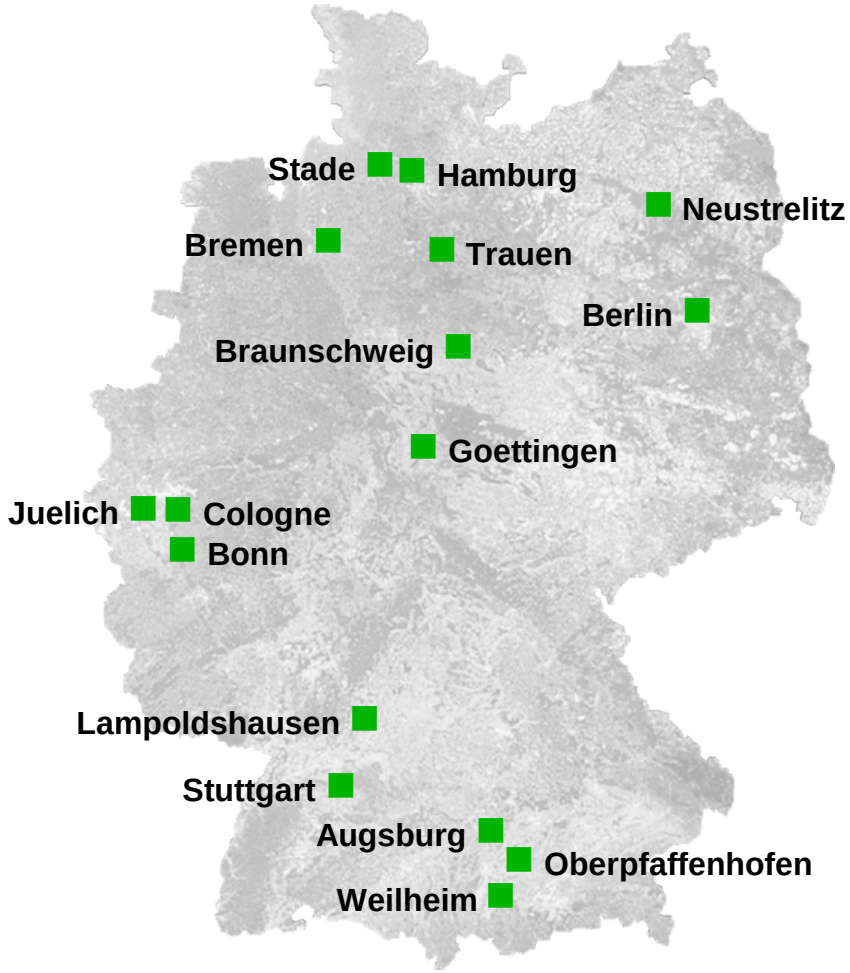


- Research Institution
- Space Agency
- Project Management Agency

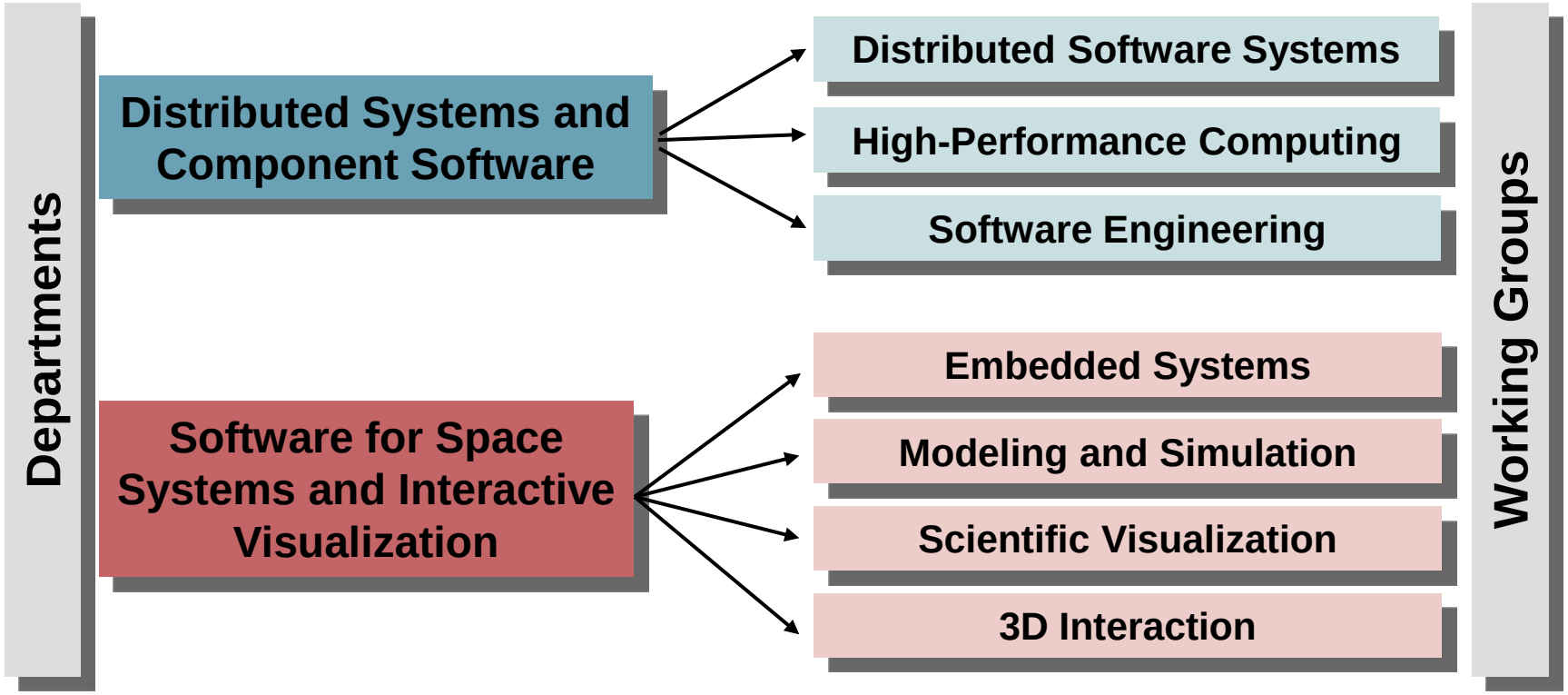
DLR Locations and Employees

Approx. 8000 employees across 33 institutes and facilities at
- 16 sites.

Offices in Brussels, Paris, Tokyo and Washington.



DLR Institute Simulation and Software Technology
Scientific Themes and Working Groups



Survey

- ESSEX motivation
- The ESSEX software infrastructure
- Holistic view: application, algorithm and performance
- Algorithmic developments: JADA, FEAST, CARP-CG
- Application results
- Conclusions
- The Future: ESSEX II

ESSEX Motivation: Requirements for Exascale

Hardware
Fault tolerance
Energy efficiency
New levels of parallelism

Quantum Physics Applications
Extremely large sparse matrices:
eigenvalues, spectral properties,
time evolution

ESSEX

Exascale Sparse Solver Repository (ESSR)
ghost / PHIST

FT concepts, programming for extreme parallelism

Sparse eigensolvers, preconditioners, spectral methods

Quantum physics / chemistry

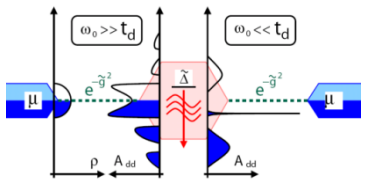
ESSEX applications: Graphene, topological insulators, …

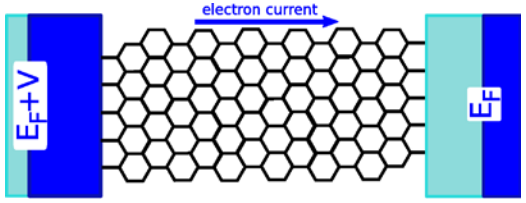
ESSEX: Physical Motivation and Sparse Eigenvalue problem

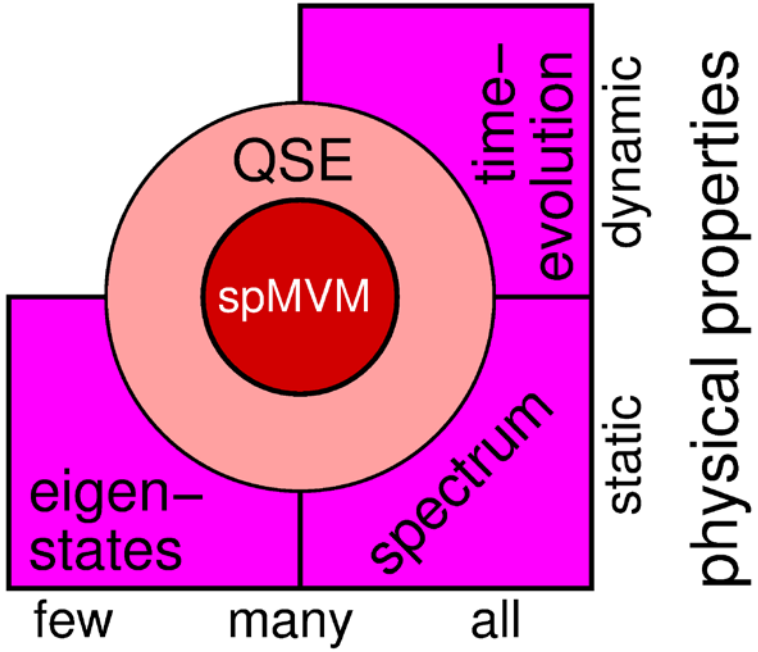
Solve large sparse eigenvalue problem

$$H\,x = \lambda\,x$$

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







(λ_i, x_i)

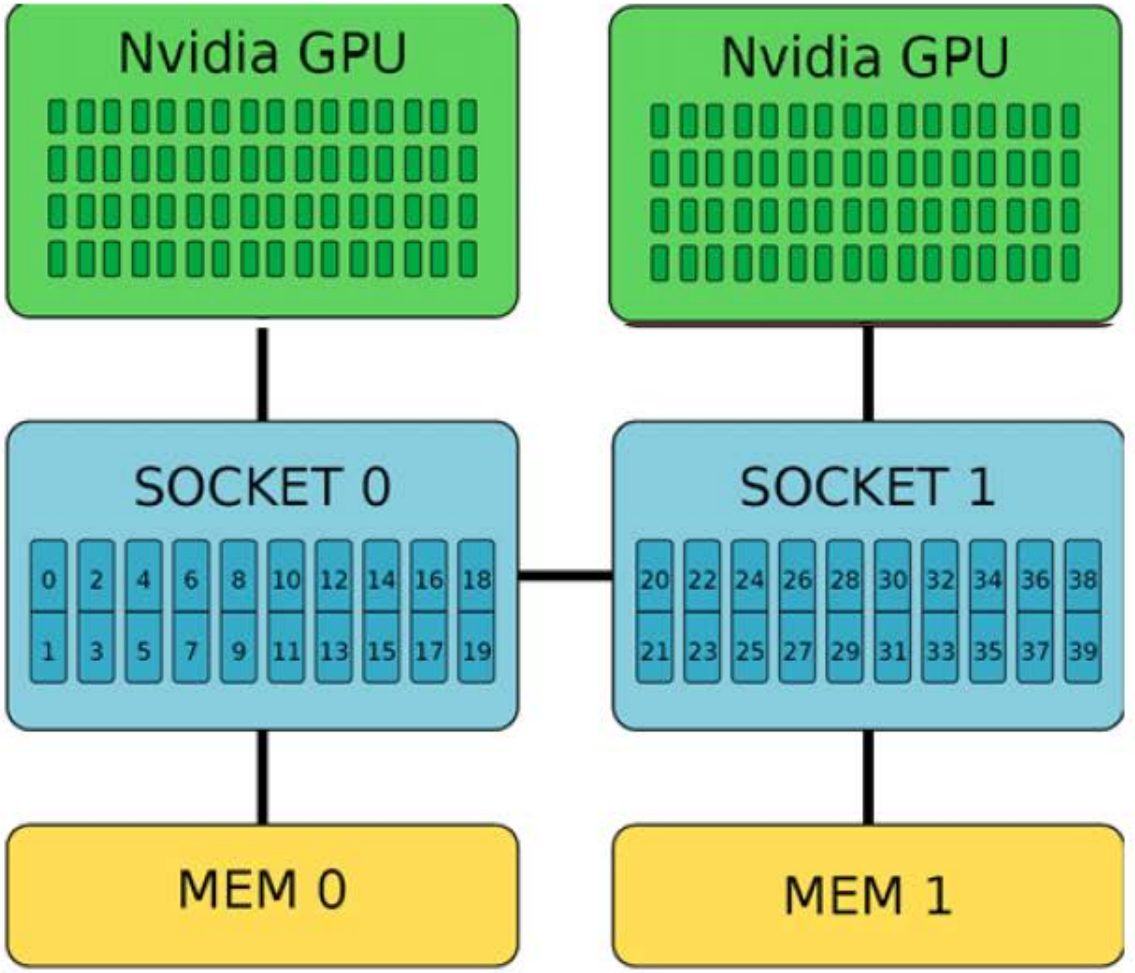
The ESSEX Software Infrastructure: Kernel Library GHOST

(General Hybrid and Optimized Sparse Toolkit) provides

- intelligent resource management for heterogenous systems
  - automatic pinning of threads to cores
  - asynchronous execution of (larger) tasks
- some fully optimized kernels for sparse matrix methods
  - sparse matrix-(multi)vector multiplication (spM(M)VM)
  - 'tall and skinny' matrices in row or column major ordering
- target platforms right now: Intel CPUs, Xeon Phi and Nvidia GPUs
- programming model: 'MPI+X',
  with X=SIMD intrinsics, OpenMP and CUDA

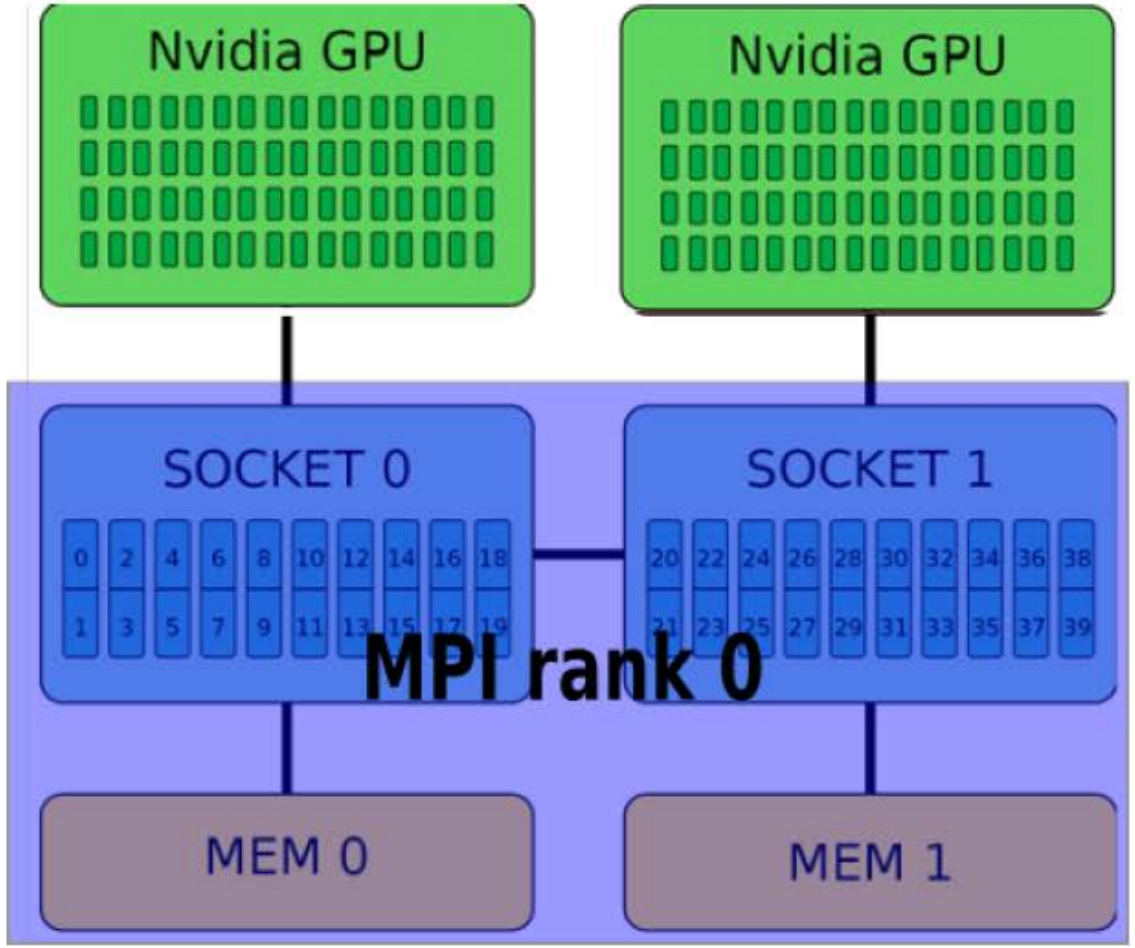
The ESSEX Software Infrastructure: MPI + X with GHOST

- System with multiple CPUs (NUMA domains) and GPUs



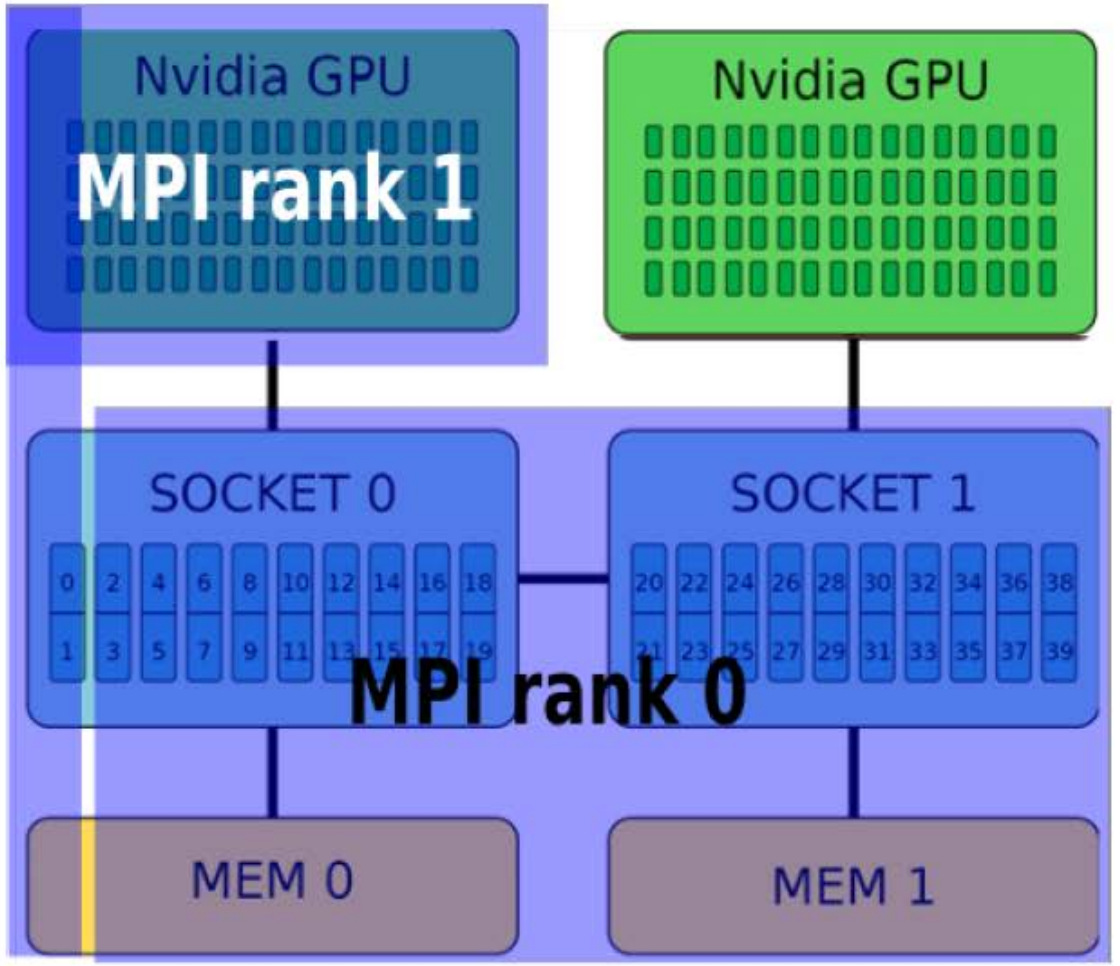
The ESSEX Software Infrastructure: MPI + X with GHOST

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



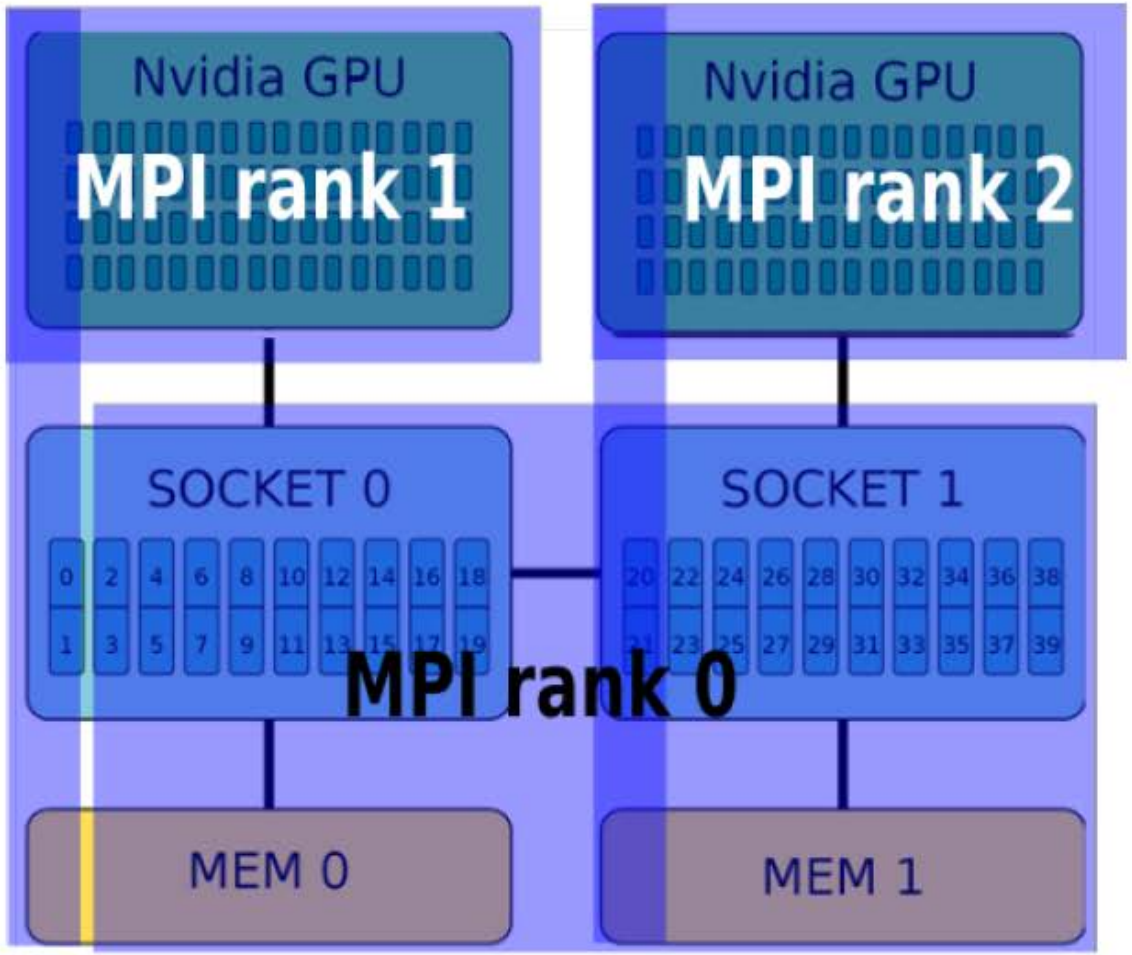
The ESSEX Software Infrastructure: MPI + X with GHOST

- 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

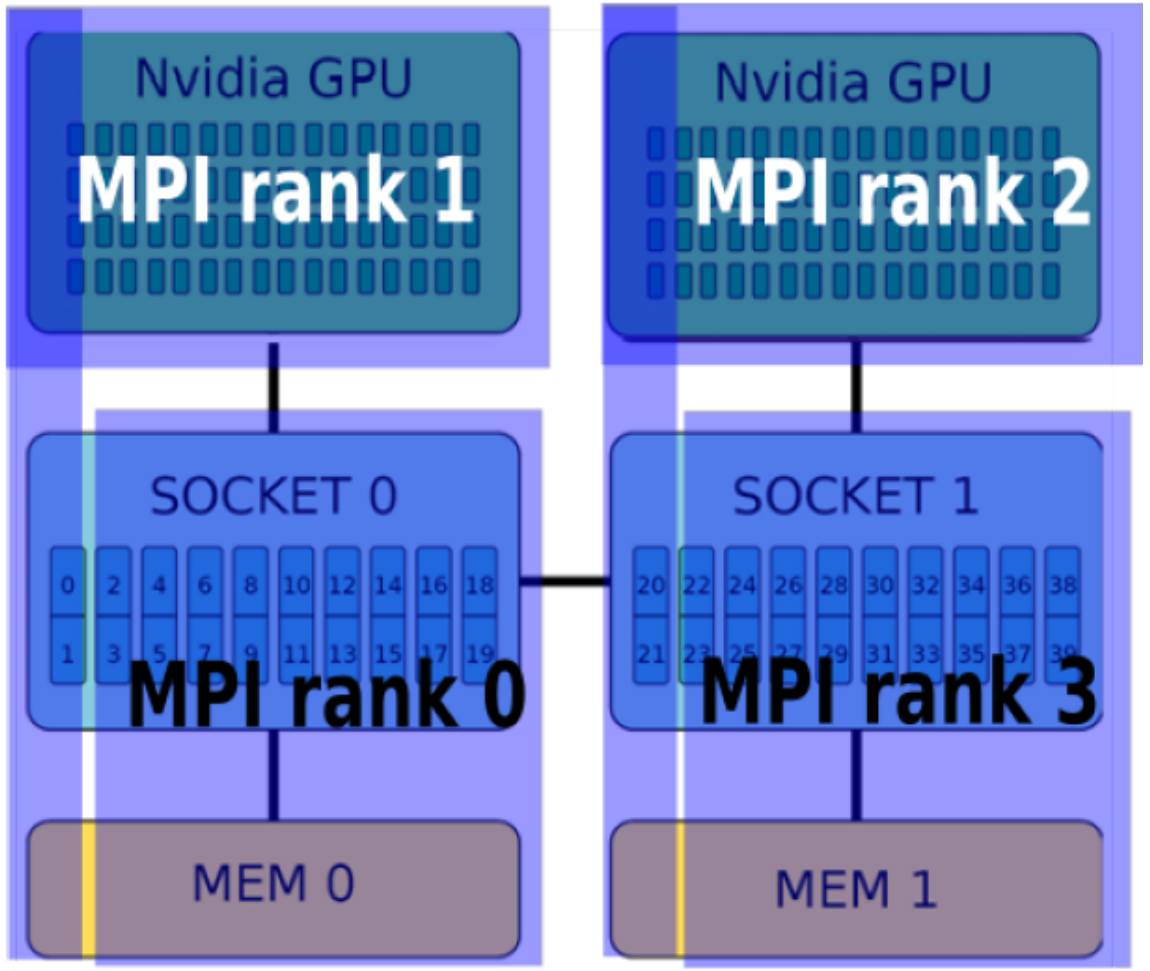
- System with multiple CPUs (NUMA domains) and GPUs
- -np 1: use entire CPU
- -np 2: use CPU and first GPU
- -np 3: use CPU and both GPUs



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

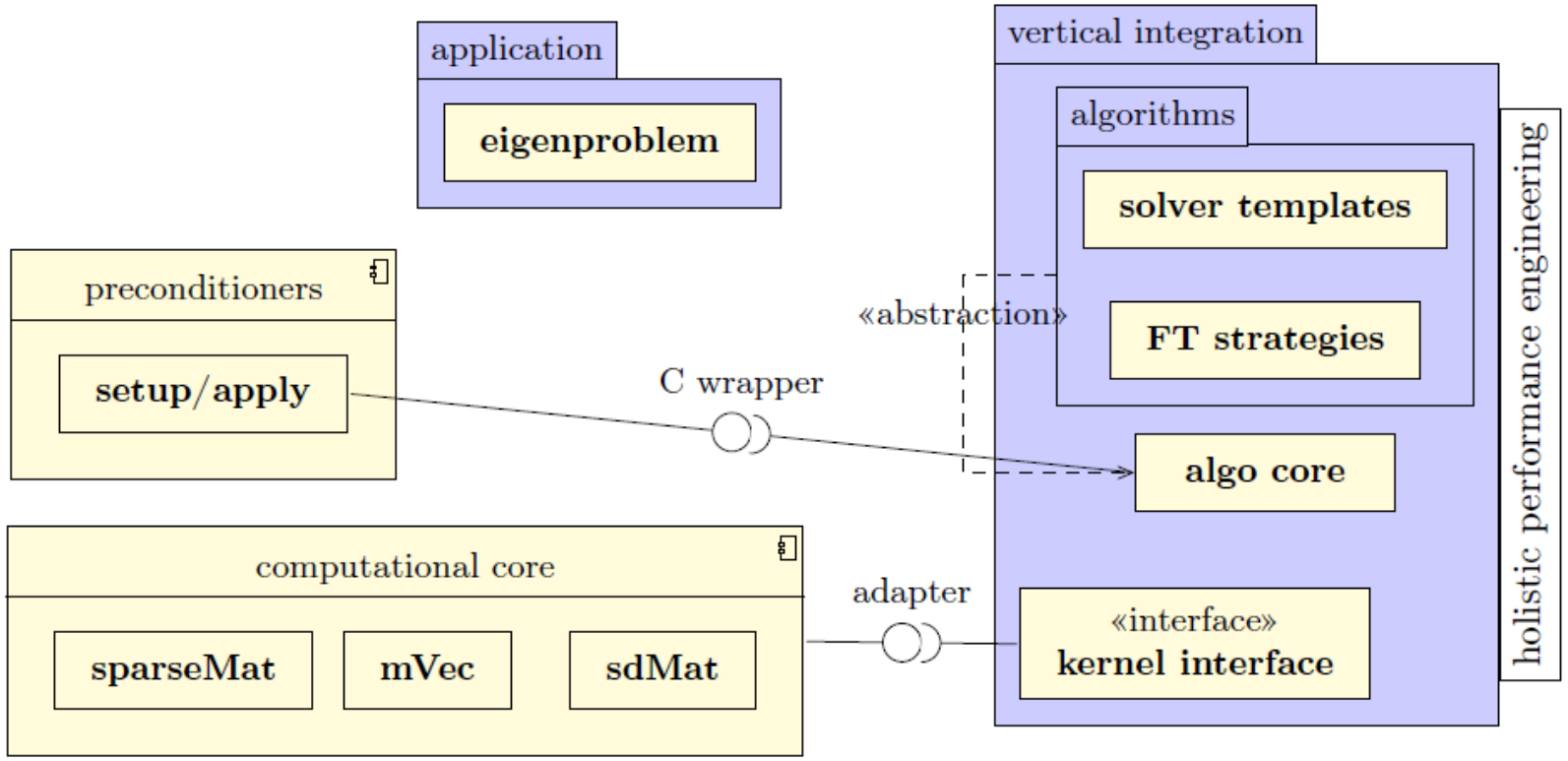
Option: distribute problem according to memory bandwidth measured



The ESSEX Software Infrastructure: PHIST for Implementing Iterative Solvers

a Pipelined Hybrid-parallel Iterative Solver Toolkit

- facilitate algorithm development using GHOST
- holistic performance engineering
- portability and interoperability

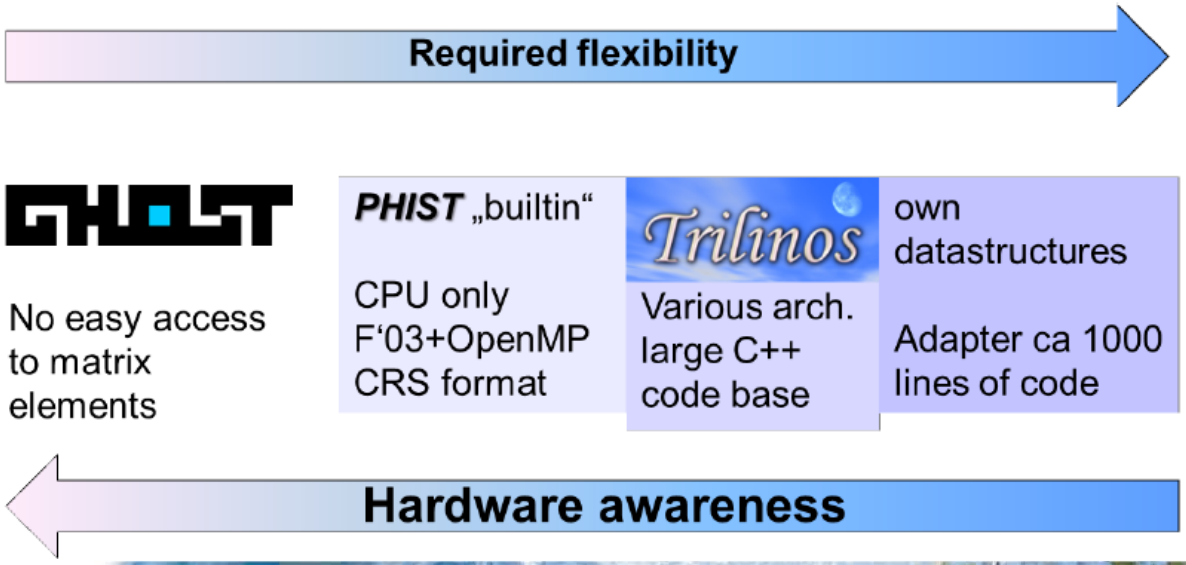


The ESSEX Software Infrastructure: PHIST for Implementing Iterative Solvers

Useful abstraction: kernel interface

Choose from several 'backends' at compile time, to

- easily use **PHIST** in existing applications
- perform the same run with different kernel libraries
- compare numerical accuracy and performance
- exploit unique features of a kernel library (e.g. preconditioners)



The ESSEX Software Infrastructure: PHIST for Implementing Iterative Solvers

Cool features of PHIST

Task macros

out-of-order execution of code blocks

- overlap comm. and comp.
- asynchronous checkpointing
- ...

Consistent random vectors

make **PHIST** runs comparable

- across platforms (CPU, GPU...)
- across kernel libraries
- independent of #procs, #threads

PerfCheck:

print achieved roofline performance of kernels after complete run to reveal

- deficiencies of kernel lib
- implementation issues of algorithm (strided data access etc.)

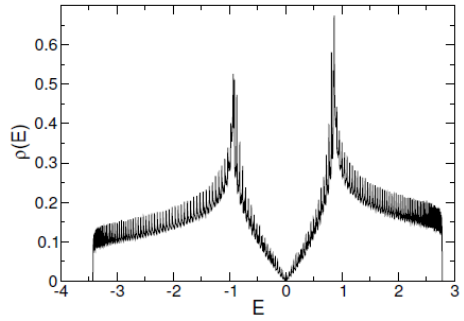
Special-purpose operations

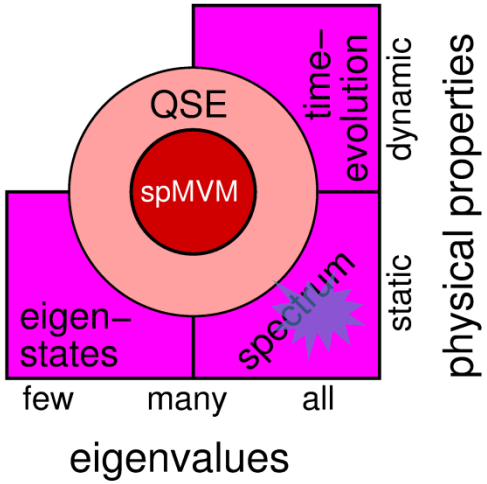
- fused kernels, e.g. compute $Y = \alpha AX + \beta Y$ *and* $Y^T X$
- highly accurate core functions, e.g. block orthogonalization in simulated quad precision

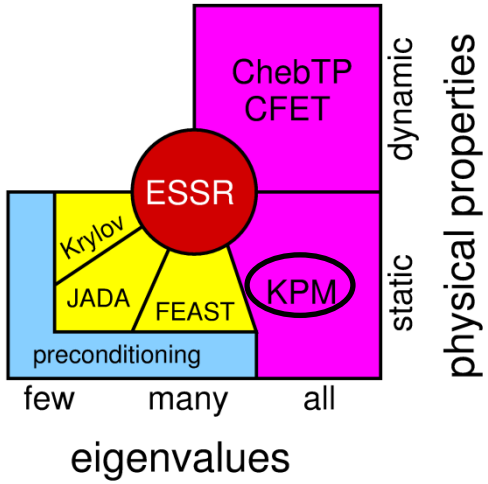
Application, Algorithm and Performance: Kernel Polynomial Method (KPM) – A Holistic View

- 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:

```
for r = 0 to R − 1 do
    |v⟩ ← |rand()⟩
    Initialization steps and computation of η0, η1
    for m = 1 to M/2 do
        swap(|w⟩, |v⟩)
        |u⟩    ← H|v⟩
        |u⟩    ← |u⟩ − b|v⟩
        |w⟩    ← −|w⟩
        |w⟩    ← |w⟩ + 2a|u⟩
        η2m    ← ⟨v|v⟩
        η2m+1  ← ⟨w|v⟩
    end for
end for
```

Application:
Loop over random initial states

Algorithm:
Loop over moments

Building blocks:
(Sparse) linear algebra library

- ▷ `spmv()` Sparse matrix vector multiply
- ▷ `axpy()` Scaled vector addition
- ▷ `scal()` Vector scale
- ▷ `axpy()` Scaled vector addition
- ▷ `nrm2()` Vector norm
- ▷ `dot()` Dot Product

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⟩ ← |rand()⟩
    Initialization steps and computation of η0, η1
    for m = 1 to M/2 do
        swap(|w⟩, |v⟩)
        |u⟩    ← H|v⟩                ▷ spmv()
        |u⟩    ← |u⟩ − b|v⟩          ▷ axpy()
        |w⟩    ← −|w⟩                ▷ scal()
        |w⟩    ← |w⟩ + 2a|u⟩         ▷ axpy()
        η2m    ← ⟨v|v⟩               ▷ nrm2()
        η2m+1  ← ⟨w|v⟩                 ▷ dot()
    end for
end for
```

```
for r = 0 to R − 1 do
    |v⟩ ← |rand()⟩
    Initialization steps and computation of η0, η1
    for m = 1 to M/2 do
        swap(|w⟩, |v⟩)
        |w⟩ = 2a(H − b𝟙)|v⟩ − |w⟩ &
            η2m   = ⟨v|v⟩ &
            η2m+1 = ⟨w|v⟩          ▷ aug_spmv()
    end for
```

Augmented Sparse Matrix Vector Multiply

The Kernel Polynomial Method (KPM)

Optimal performance exploit knowledge from all software layers!

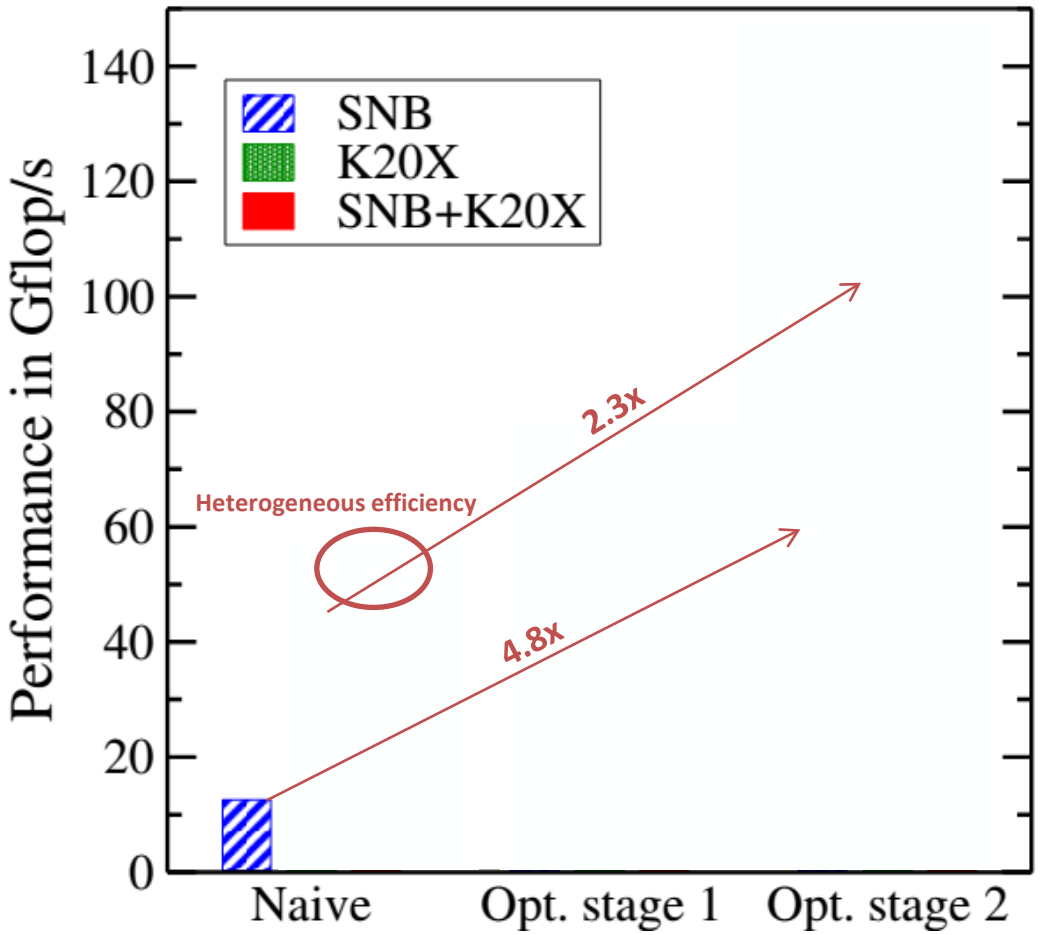
Basic algorithm – Compute Cheyshev polynomials/moments:

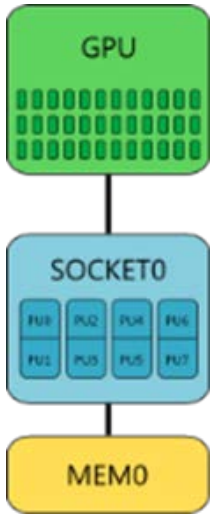
```
for r = 0 to R − 1 do
    |v⟩ ← |rand()⟩
    Initialization steps and computation of η0, η1
    for m = 1 to M/2 do
        swap(|w⟩, |v⟩)
        |w⟩ = 2a(H − b1)|v⟩ − |w⟩ &
            η2m   = ⟨v|v⟩ &
            η2m+1 = ⟨w|v⟩                    ▷ aug_spmv()
    end for
```

```
|V⟩ := |v⟩0..R−1                          ▷ Assemble vector blocks
|W⟩ := |w⟩0..R−1
|V⟩ ← |rand()⟩
Initialization steps and computation of μ0, μ1
for m = 1 to M/2 do
    swap(|W⟩, |V⟩)
    |W⟩ = 2a(H − b1)|V⟩ − |W⟩ &
            η2m[:]   = ⟨V|V⟩ &
            η2m+1[:] = ⟨W|V⟩                 ▷ aug_spmmv()
end for
```

Augmented Sparse Matrix Multiple Vector Multiply

KPM: Heterogenous Node Performance



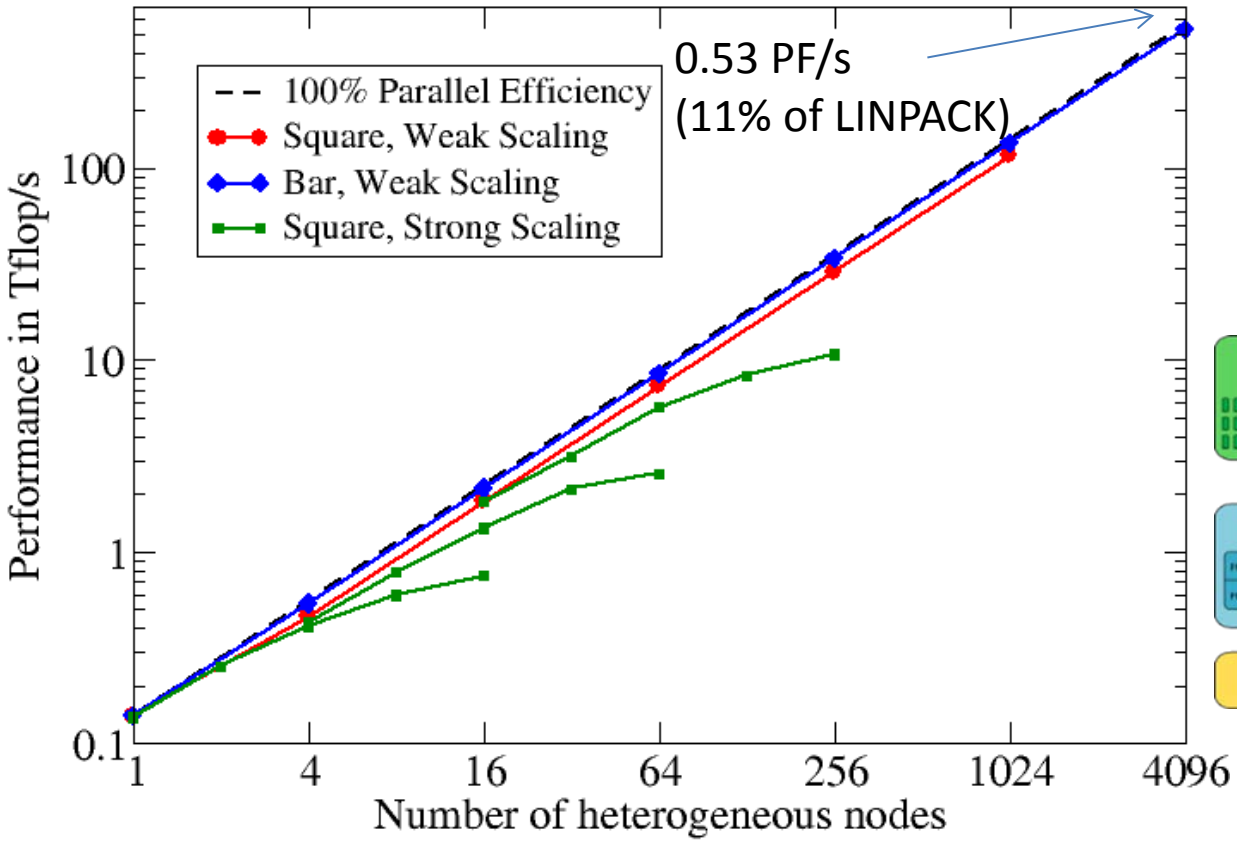


NVIDIDA K20X

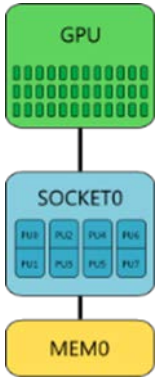
Intel
Xeon E5-2670 (SNB)

- 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

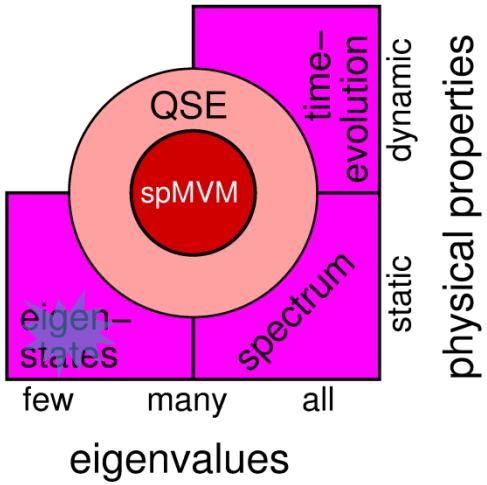
Performance Engineering of the Kernel Polynomial Method on Large-Scale CPU-GPU Systems
M. Kreutzer, A. Pieper, G. Hager, A. Alvermann, G. Wellein and H. Fehske, IEEE IPDPS 2015

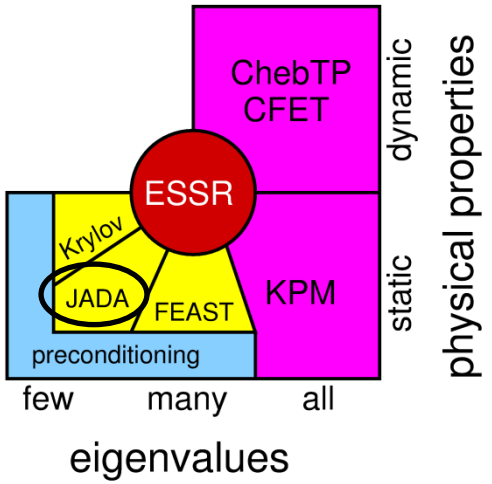
*Thanks to CSCS/T. Schulthess for granting access and compute time

Algorithmic Developments: Blocked Jacobi-Davidson (JADA) Method

Compute l **extreme eigenvalues/-vectors** $\{(\lambda_i, v_i), i = 1, \ldots, l\}$
of sparse matrix A:

$$A\, v_i = \lambda_i v_i$$





Algorithmic Developments: Blocked JADA – exploit benefit of block spMVM

Blocked JADA method: Solve n_b correction equations at the same time.
Basic BLOCKED JADA operator becomes (j=1,…,n_b):

$$(y_j \leftarrow (I - QQ^T)(A - \tau_j I)x_j)$$

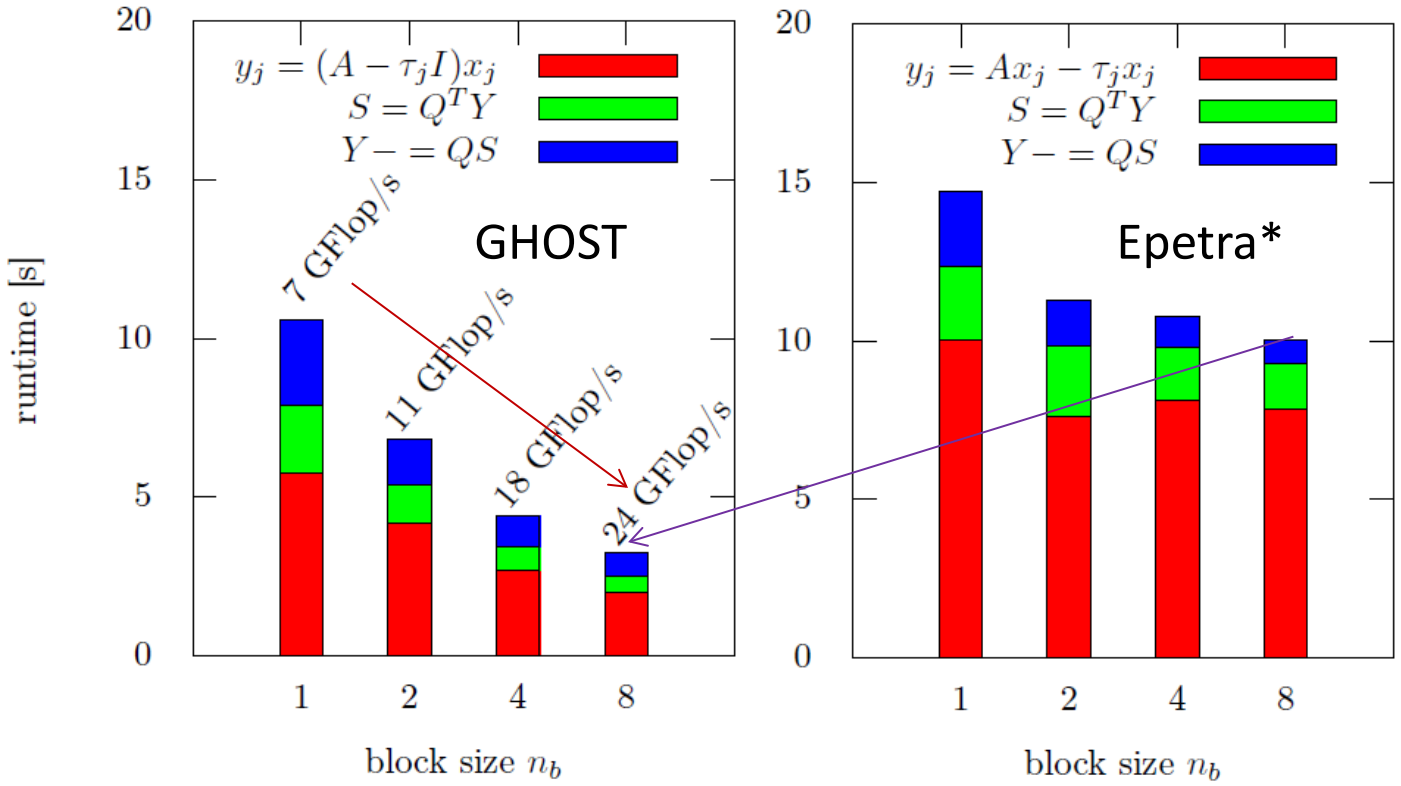
Scalar

Dense matrix (Tall & skiny)

Sparse Matrix

BLOCKED JADA operation available in GHOST for CPU - GPGPU & Xeon Phi: work in progress.

Algorithmic Developments: Blocked JADA – performance of basic operation



Matrix:
$D=10^7$; $n_{nzr}=14$
Intel Xeon E5-2660 v2
120 JADA operations

3.3x over-compensates numerical overhead of blocking!

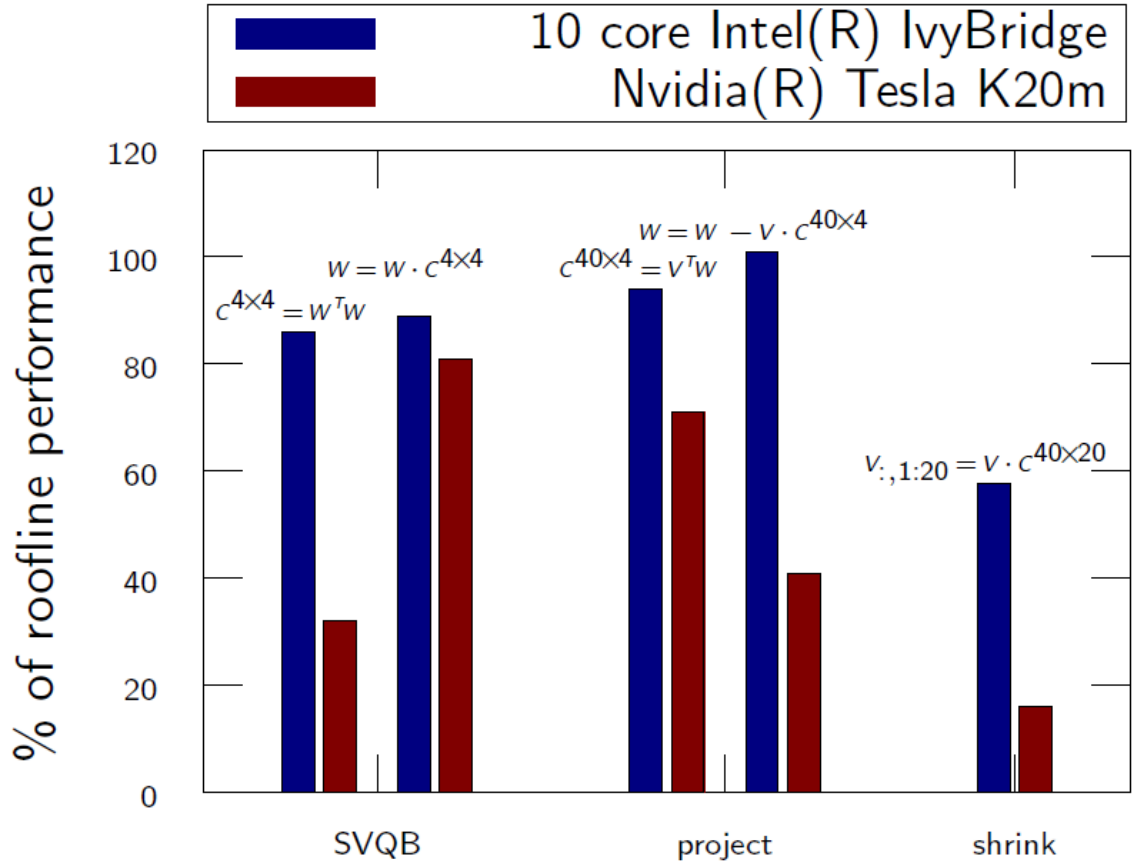
2.5x vs. Trilinos building blocks

Increasing the Performance of the Jacobi-Davidson Method by blocking
M. Röhrig-Zöllner, J. Thies, A. Basermann et al., SIAM SISC 2015.

*http://trilinos.sandia.gov/packages/epetra/

Algorithmic Developments: Blocked JADA – `tall and skinny´ kernels

'Tall & skinny' kernel performance ($V \in \mathbb{R}^{10M \times 40}$, $W \in \mathbb{R}^{10M \times 4}$)



⇒ some fallback kernels needed on GPU, further experiments postponed

Algorithmic Developments: Blocked JADA – total algorithm

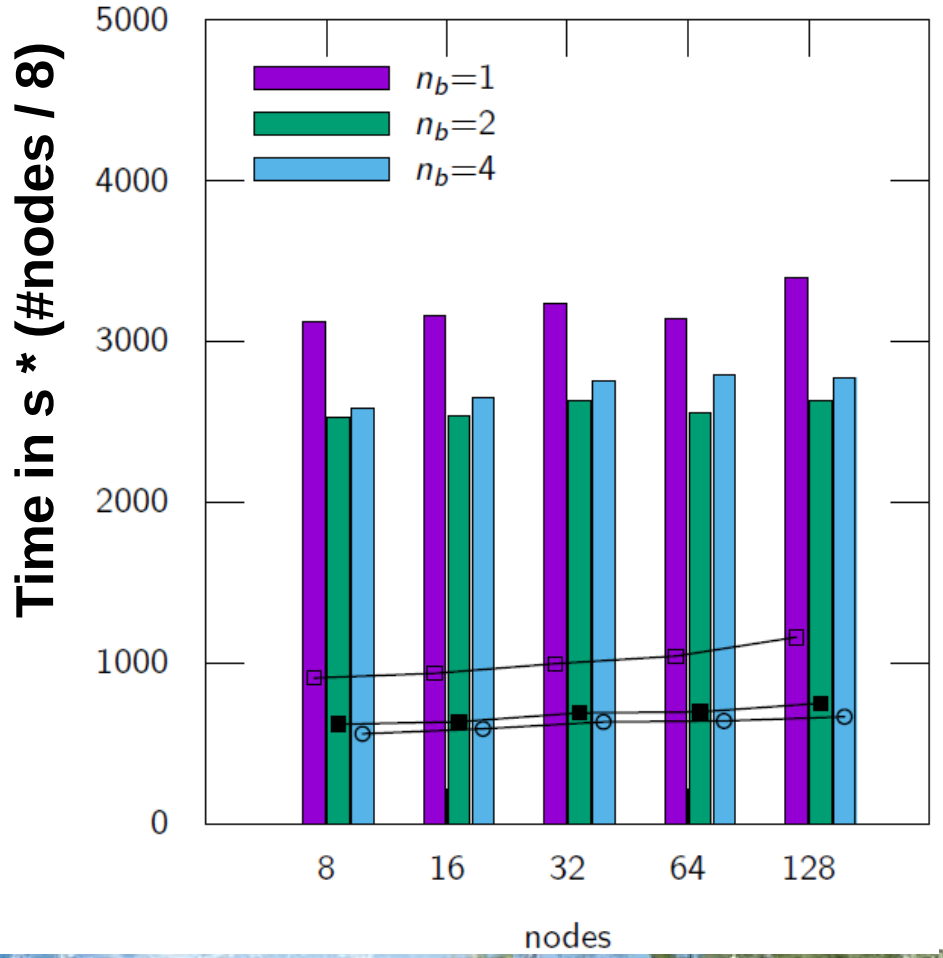
Strong scaling performance

Setup

- non-symmetric matrix from 7-point 3D PDE discretization ($n \approx 1.3 \cdot 10^8$, $n_{nz} \approx 9.4 \cdot 10^8$)
- find 20 eigenvalues
- Ivy Bridge Cluster

Results

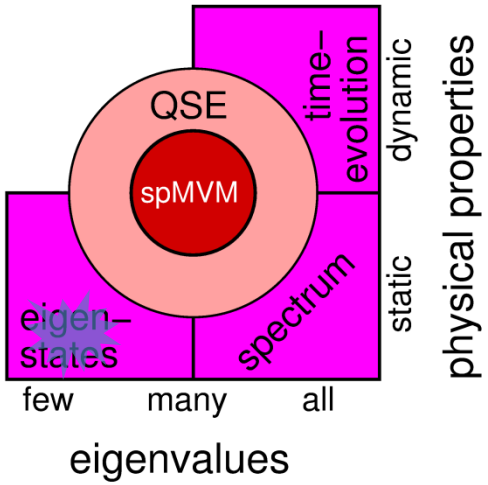
- $n_b = 2$: significantly faster
- $n_b = 4$: no further improvement

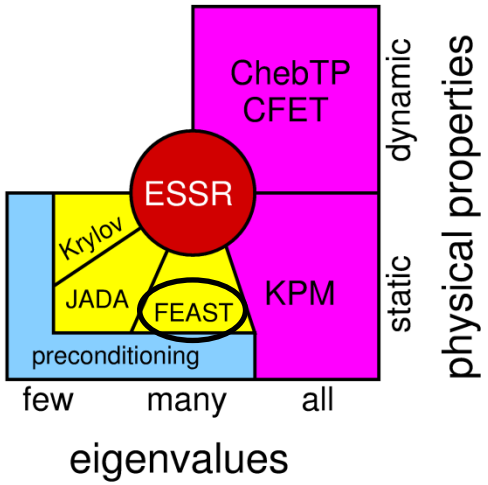


Algorithmic Developments: FEAST method and CARP-CG solver

Compute l **interior eigenvalues/-vectors** $\{(\lambda_i, v_i), i = 1, \ldots, l\}$
of sparse matrix A:

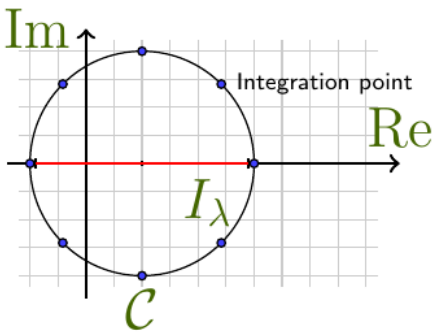
$$A\, v_i = \lambda_i v_i$$





Algorithmic Developments: FEAST – Progress towards Large Scale

- FEAST = Numerical integration + Rayleigh-Ritz
- Eigenvalues in given **interval**
- Numerical Integration:
  Solution of **many large** linear systems



Achievements:

- Estimation of eigenvalue count (also with **KPM**)
- Integration of linear solver **CARP-CG**
- Graphene eigenvalue problems
- Substitution of linear solver by **polynomials**

On the parallel iterative solution of linear systems arising in the FEAST algorithm for computing inner eigenvalues
J. Thies, A. Basermann, B. Lang et al.:
Parallel Computing 49 (2015) 153–163

→ Few inner eigenvalues of graphene problem of size **10^8**

Compare with state of the art FEAST: 10^5 using direct sparse solver

Algorithmic Developments: CARP-CG Preconditioner for Inner Eigenproblems

FEAST eigensolver yields challenging linear systems
(indefinite, random entries, small diagonal elements)

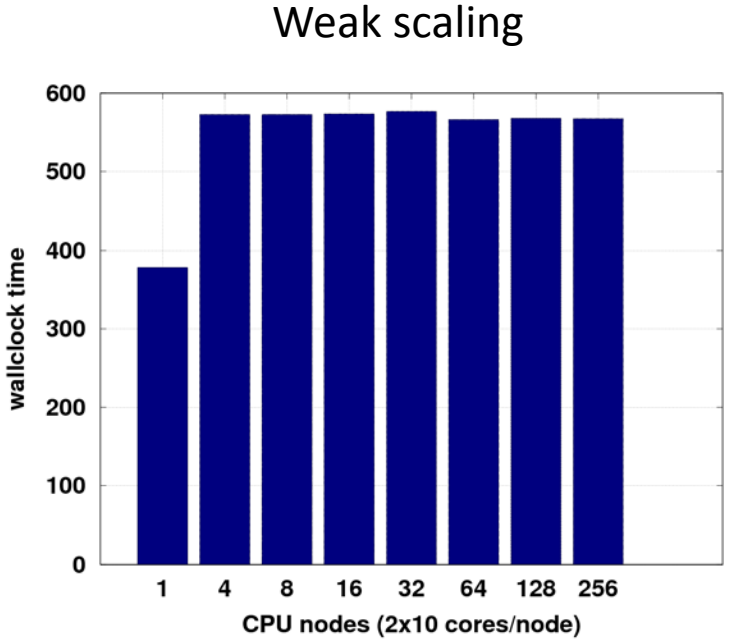
CARP-CG: a Conjugate Gradient accelerated Kaczmarz method

- Numerically very robust
- Sparse kernel: successive row projections ($a_{k,:}$ is the k'th row of A)

$$x_{k+1} \leftarrow x_k - (a_{k,:}x_k)a_{k,:}^T$$

- Data dependency resolved by node-local graph coloring
- Component averaging between nodes (recovers global Kaczmarz)

- Not yet fully optimized in GHOST

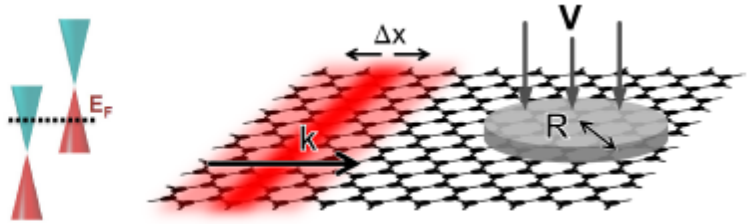


Application results

Graphene nanoribbon (GNR) with gate-defined quantum dots

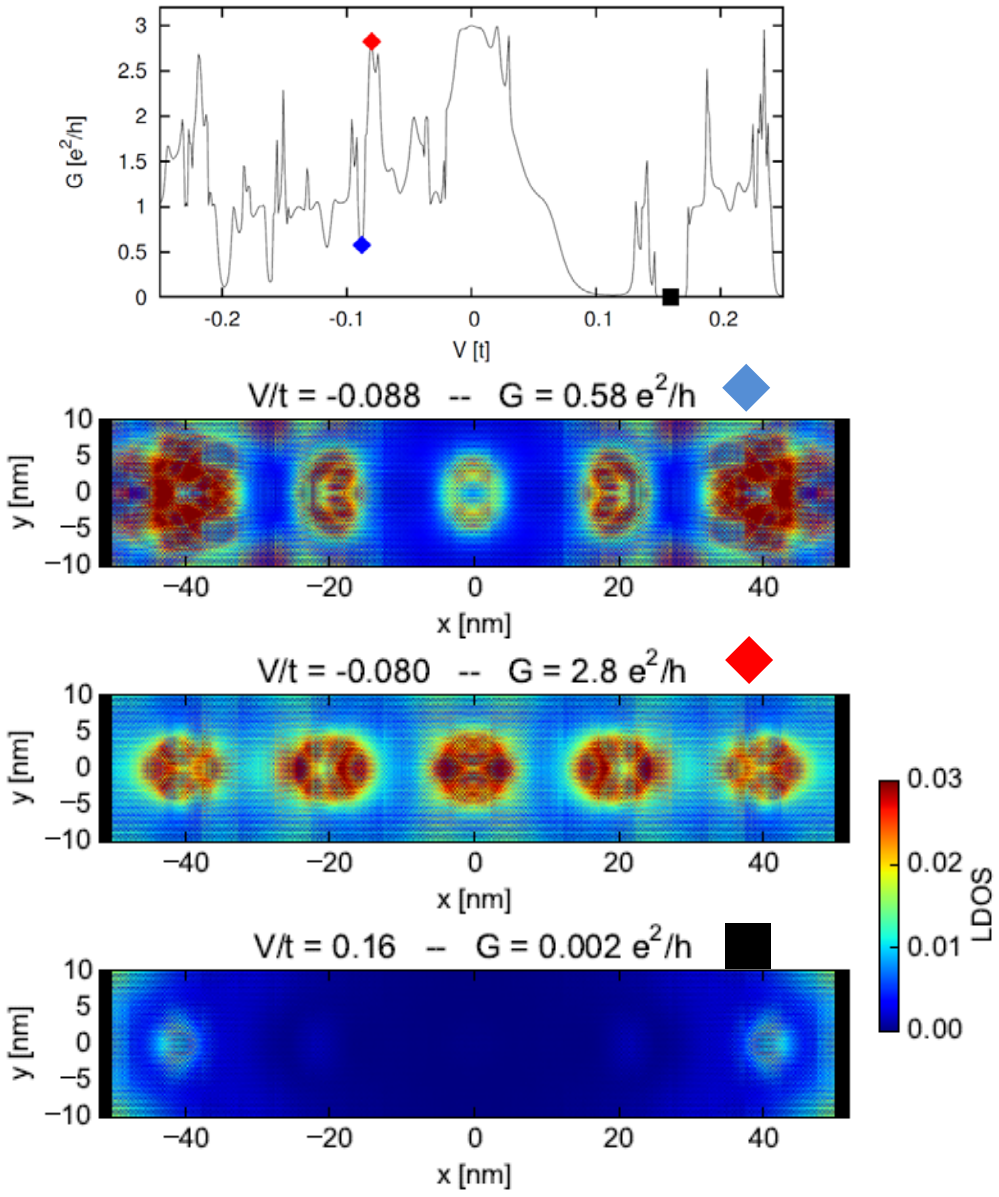
$$H = \sum_i V(i) c_i^\dagger c_i - t \sum_{\langle i,j \rangle} c_i^\dagger c_j$$

$$V(i) = V \sum_n \Theta(R - |\vec{r}_i - \vec{r}_n|)$$



Application results: GNR with 5 Gate Defined Quantum Dots

- Conductivity **G** controlled by dot potential **V/t**
- Small change in V/t
  → large change in **G**
  → GNR may realize very sensitive switch
- Superlattice – opening of band gap
  → Vanishing conductance

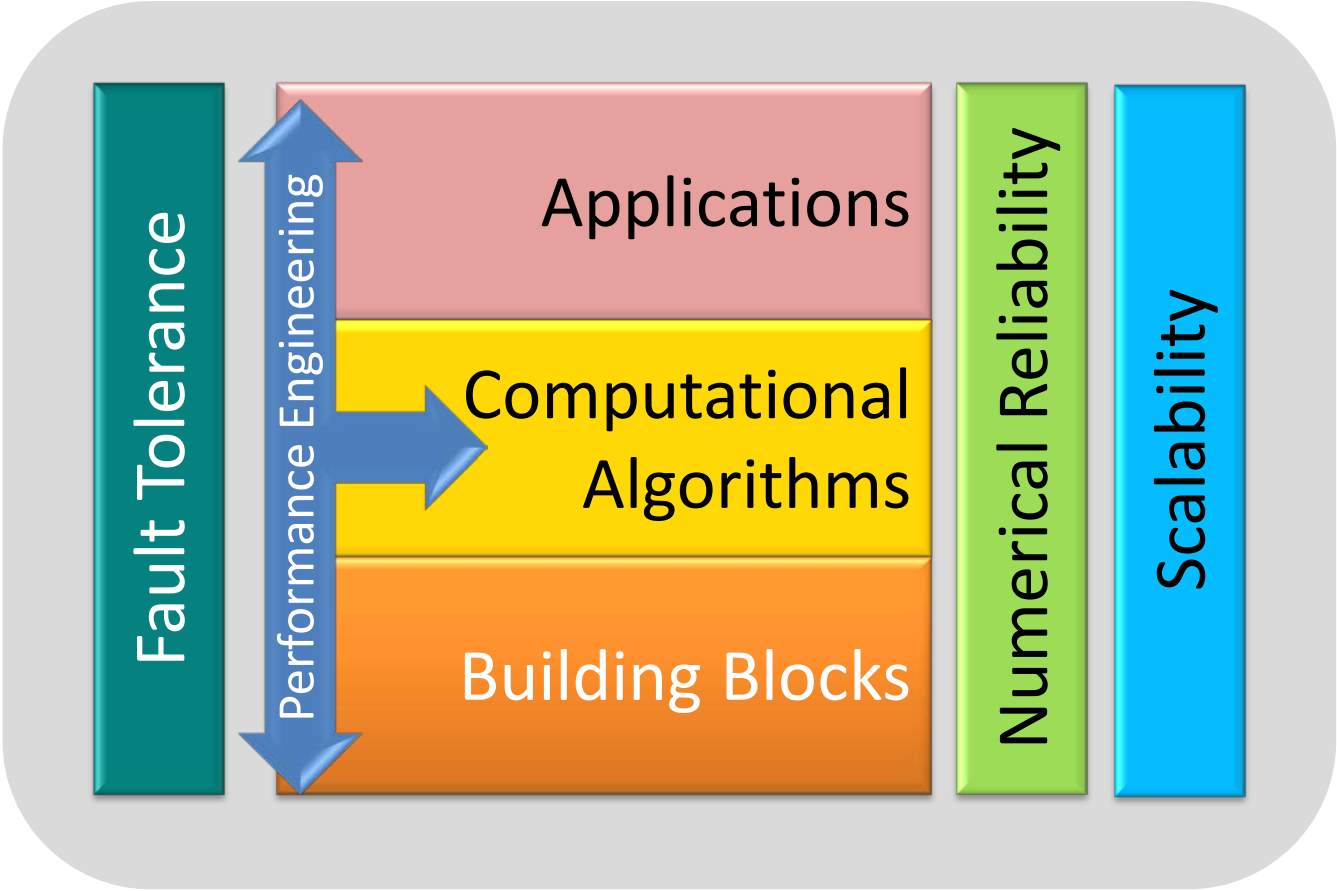


Conclusions

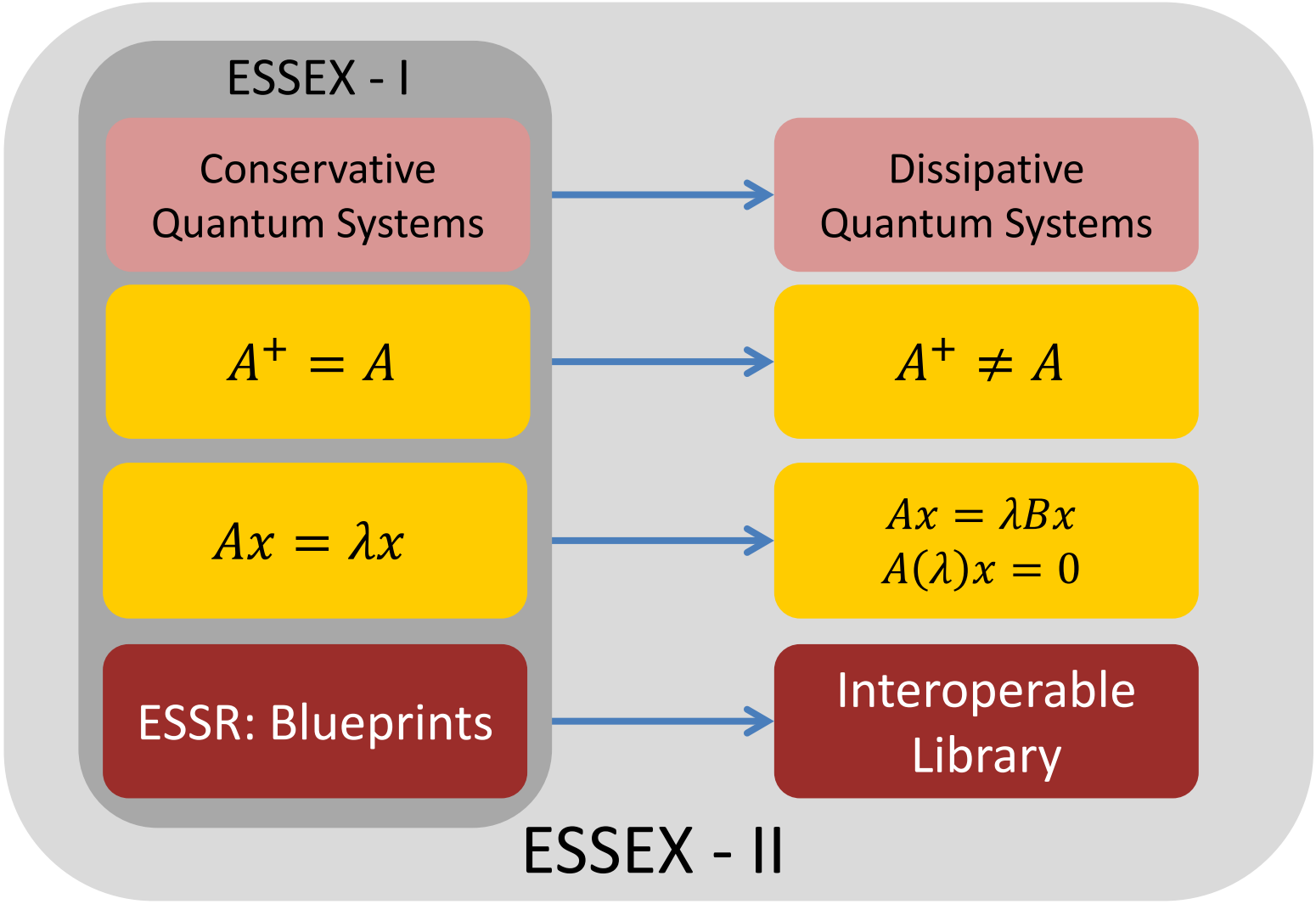
- Holistic performance engineering strategie successful for developing highly scalable solutions, cf. KPM.
- ***PHIST*** with GHOST provides a pragmatic, flexible and hardware-aware programming model for heterogeneous systems.
  - Includes highly scalable sparse iterative solvers for eigenproblems and systems of linear equations
  - Well suited for iterative solver development and solver integration into applications
- Block operations distinctly increase performance of building blocks for iterative eigensolvers like KPM or JADA.
- CARP-CG with node-level multi-coloring parallelization is suitable for robust iterative solution of nearly singular equations.
  - Appropriate iterative solver for FEAST in order to find interior eigenpairs,
  - in particular for problems from graphene design
- First convincing results with quantum physics applications

The Future: ESSEX II

- DFG confirmed ESSEX extension to 2018.
- Additional partners from Japan
  - Kengo Nakajima, Computer Science, University of Tokyo
  - Tetsuya Sakurai, Applied Mathematics, University of Tsukuba
- Main objectives
  - Enabling Exascale through software co-design
  - Established exascale sparse solver repository



Project Evolution



Thanks

Thanks to all partners from the **ESSEX** project

and to DFG for the support through the Priority Programme 1648 “Software for Exascale Computing”.

SPPEXA

Computer Science, Univ. Erlangen

Applied Computer Science, Univ. Wuppertal

Institute for Physics, Univ. Greifswald

Erlangen Regional Computing Center

- project website incl. list of publications: `http://blogs.fau.de/essex/`
- source code: `https://bitbucket.org/essex/[ghost|phist]`

International contacts

Sandia (Trilinos project)
Tenessee (Dongarra)
Japan: Tsukuba, Tokyo
The Netherlands: Groningen, Utrecht

Many thanks for your attention!

Questions?

Dr.-Ing. Achim Basermann

German Aerospace Center (DLR)

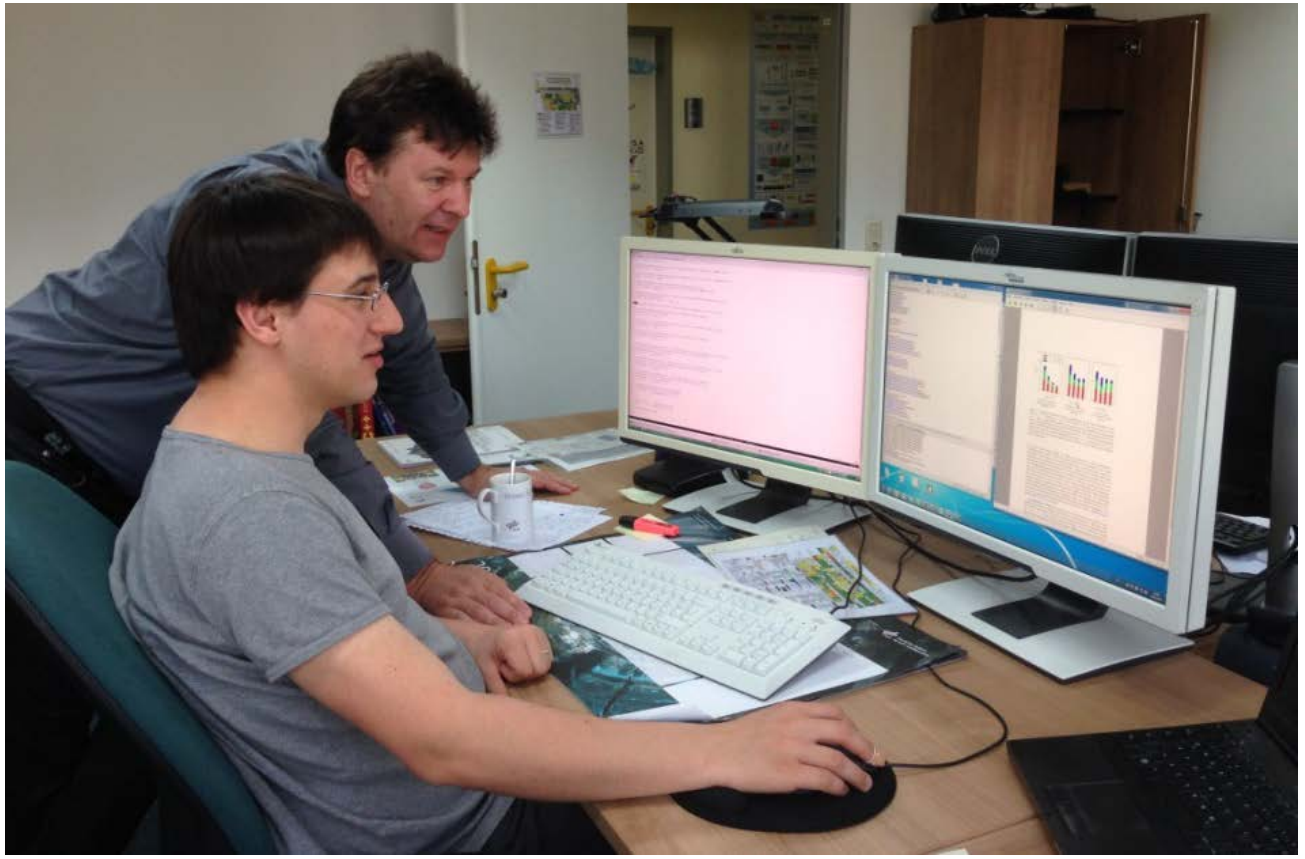
Simulation and Software Technology

Department Distributed Systems and Component Software

Team High Performance Computing

Achim.Basermann@dlr.de

http://www.DLR.de/sc

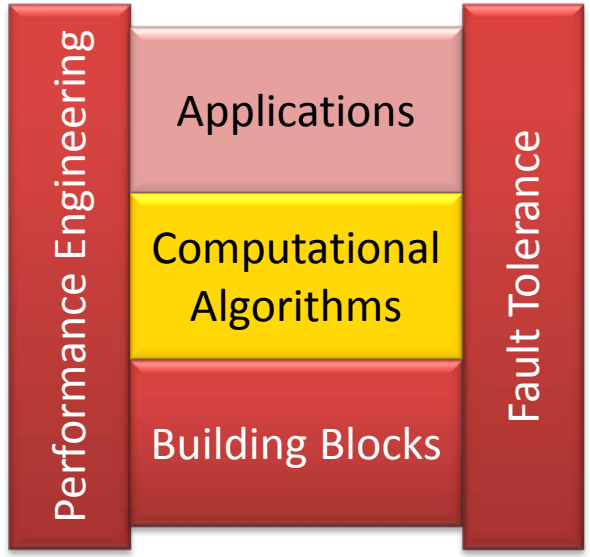


Programming

Extended

Building Blocks, Parallelization, and Performance Engineering

- Holistic performance and power engineering
- Advanced building blocks engineering



Extended

Fault Tolerance

- From prototype to application software
  - Asynchronous checkpointing & I/O
  - Automatically fault-tolerant applications

NEW

Numerical Reliability

- Performance aspects
  - Silent data corruption / skeptical programming
  - High-precision reduction operations

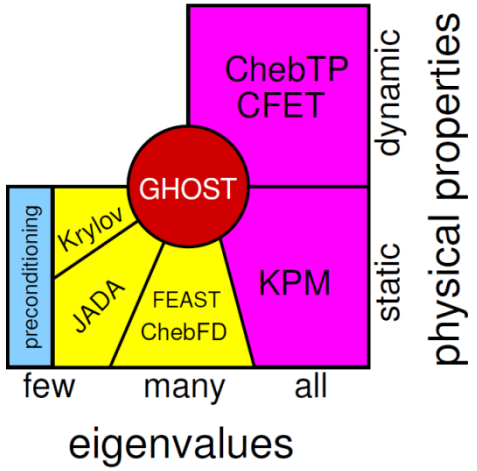
Computational Algorithms

NEW

- **Non-Hermitian**: ChebTP / CFET / JaDa
  - Extreme-scale simulations for dissipative quantum systems
  - Numerical range computation & matrix balancing

Extended

- **Cheb**yshev**F**ilter**D**iagonalization:
  - $>10^3$ interior eigenvalues of $>10^9$ matrix dim.
  - Simple, HW-efficient & low synchronization cost



NEW

- **Preconditioning** & **Communication Hiding**
  - Asynchronous JaDa: “pipelining” & preconditioning
  - AMG preconditioning for blocked JaDa & FEAST

Kengo Nakajima, University of Tokyo

NEW

- Leveraging FEAST techniques + GHOST → Nonlinear Sakurai-Sugiura Method (NSSM)

Tetsuya Sakurai, University of Tsukuba

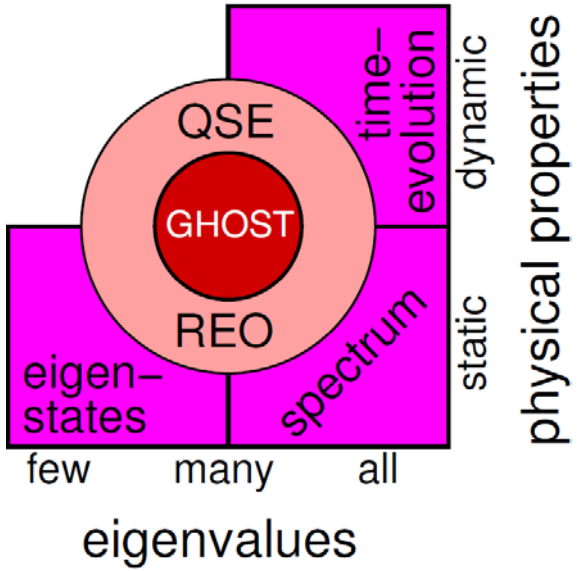
Applications

Extended

- **Quantum State Encoding (QSE)**
  - Complex (non-stencil) matrix structure encoding
  - Dissipative systems: Sparse ⊗ Dense

New

- **Matrix Reordering Strategies (REO)**
  - Application-specific
  - General techniques, e.g. **PMRSB**



Extended

- **Quantum Physics/Information Applications**
  - **Topological materials**
    Graphene & topological insulators
  - **Dissipative quantum systems**
    Light-harvesting molecules & optomechanics
  - Rich collection of quantum physics problems

$A^+ = A$

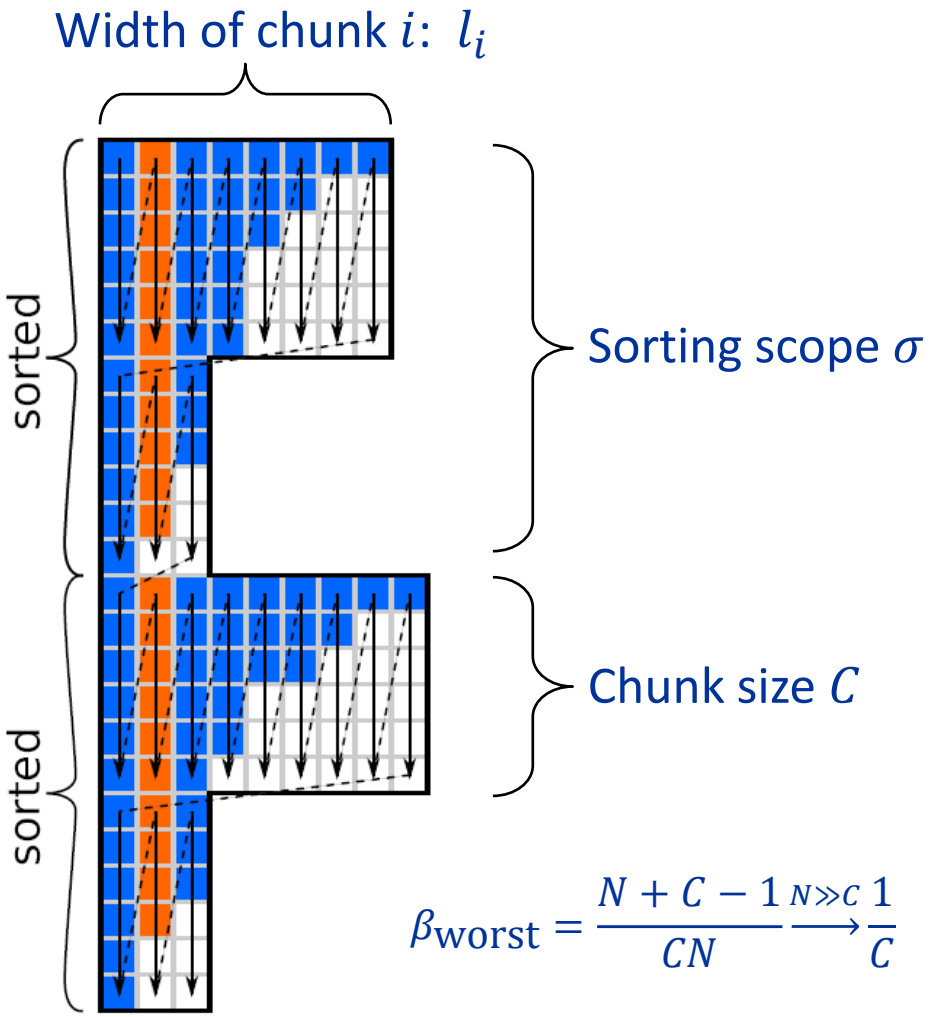
$A^+ \neq A$

Constructing SELL-C-σ

1. Pick chunk size C (guided by SIMD/T widths)
2. Pick sorting scope σ
3. Sort rows by length within each sorting scope
4. Pad chunks with zeros to make them rectangular
5. Store matrix data in “chunk column major order”

“Chunk occupancy”: fraction of “useful” matrix entries

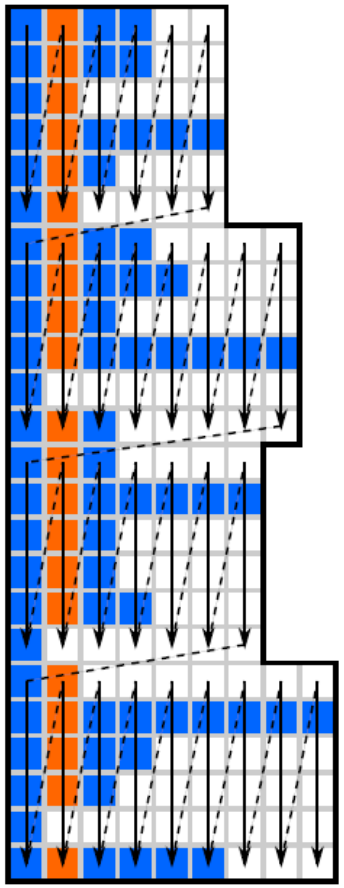
$$\beta = \frac{N_{nz}}{\sum_{i=0}^{N_c} C \cdot l_i}$$



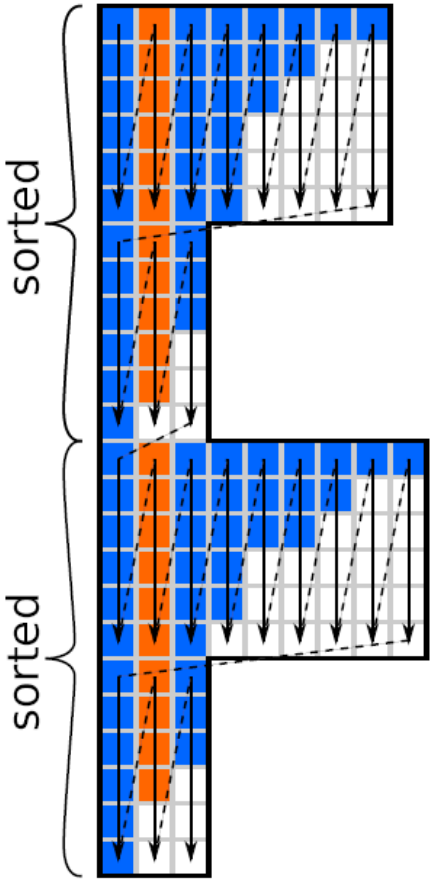
$$\beta_{\text{worst}} = \frac{N + C - 1}{CN} \overset{N \gg C}{\longrightarrow} \frac{1}{C}$$

SELL-6-12
β=0.66

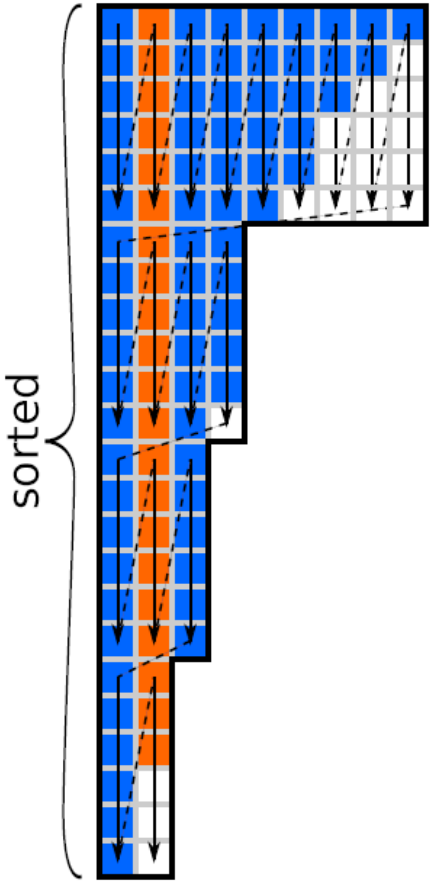
Variants of SELL-C-σ



SELL-6-1
β=0.51

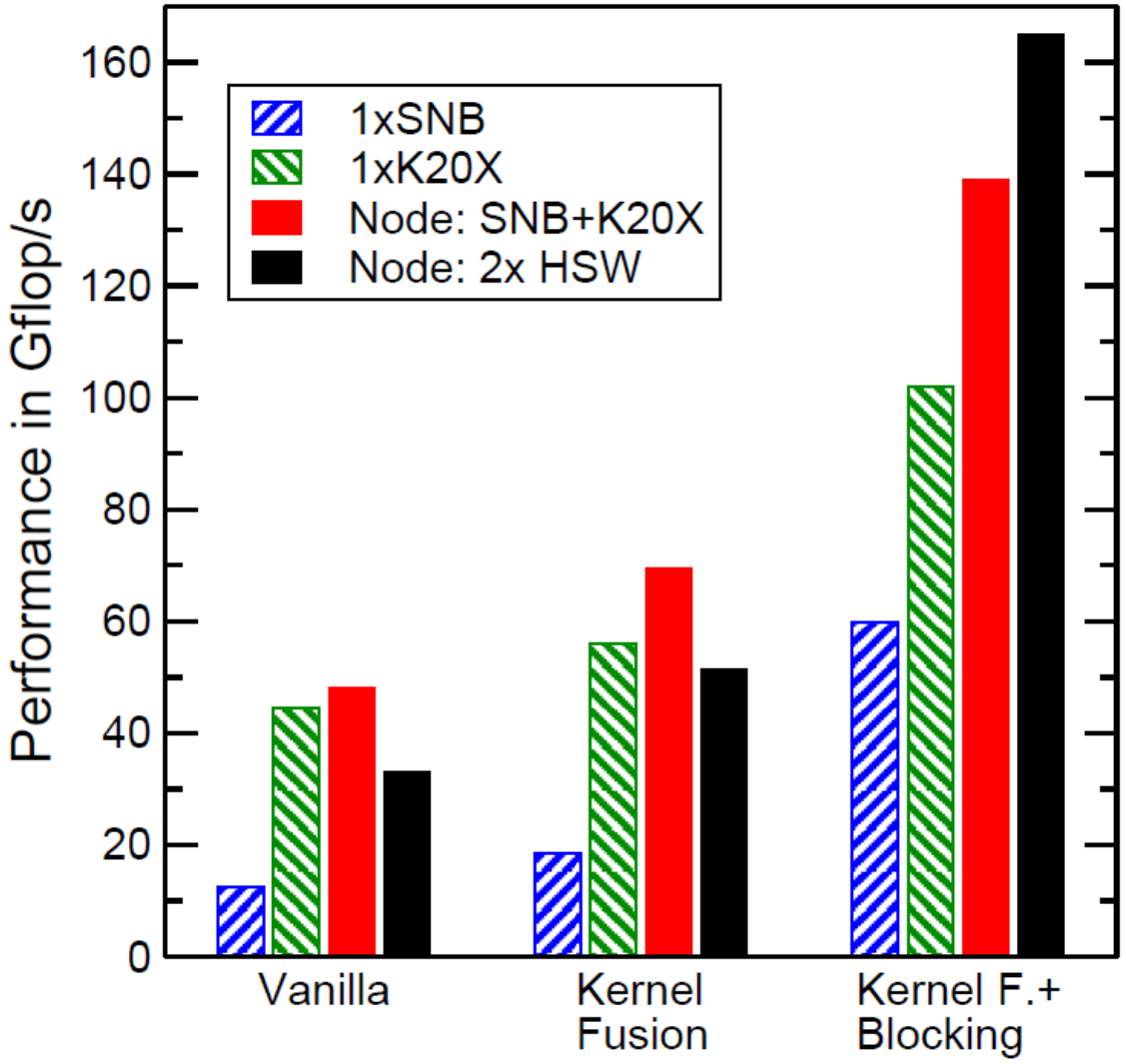


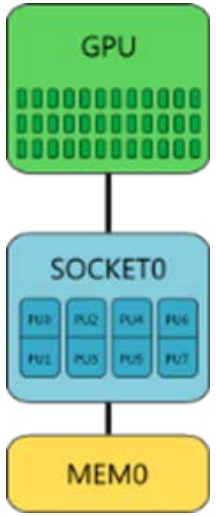
SELL-6-12
β=0.66



SELL-6-24
β=0.84

KPM: Heterogenous Node Performance





NVIDIDA K20X

Intel
Xeon E5-2670 (SNB)

2× HSW: CPU only node containing two Intel Xeon E5-2695v3 processors

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