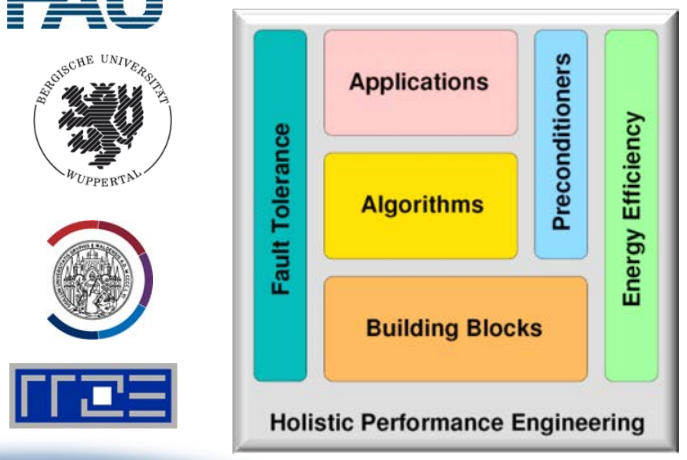
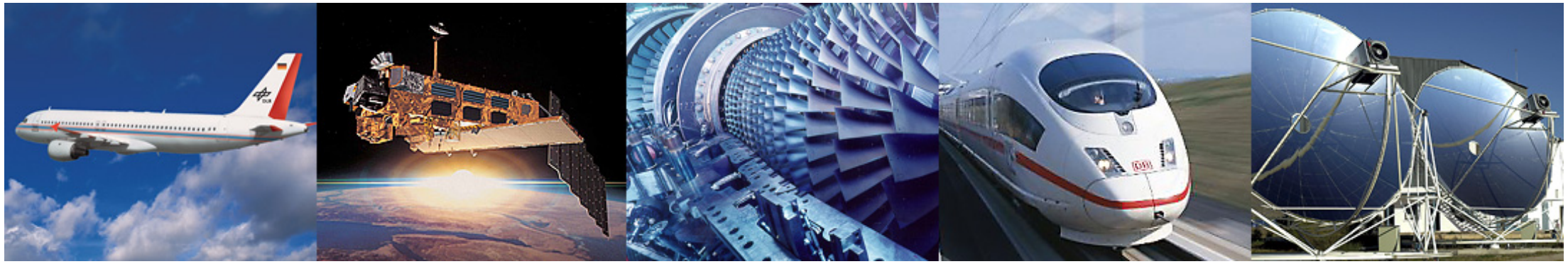
The Highly Scalable Iterative Solver Library PHIST

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

DFG Projekt ESSEX



DLR
German Aerospace Center

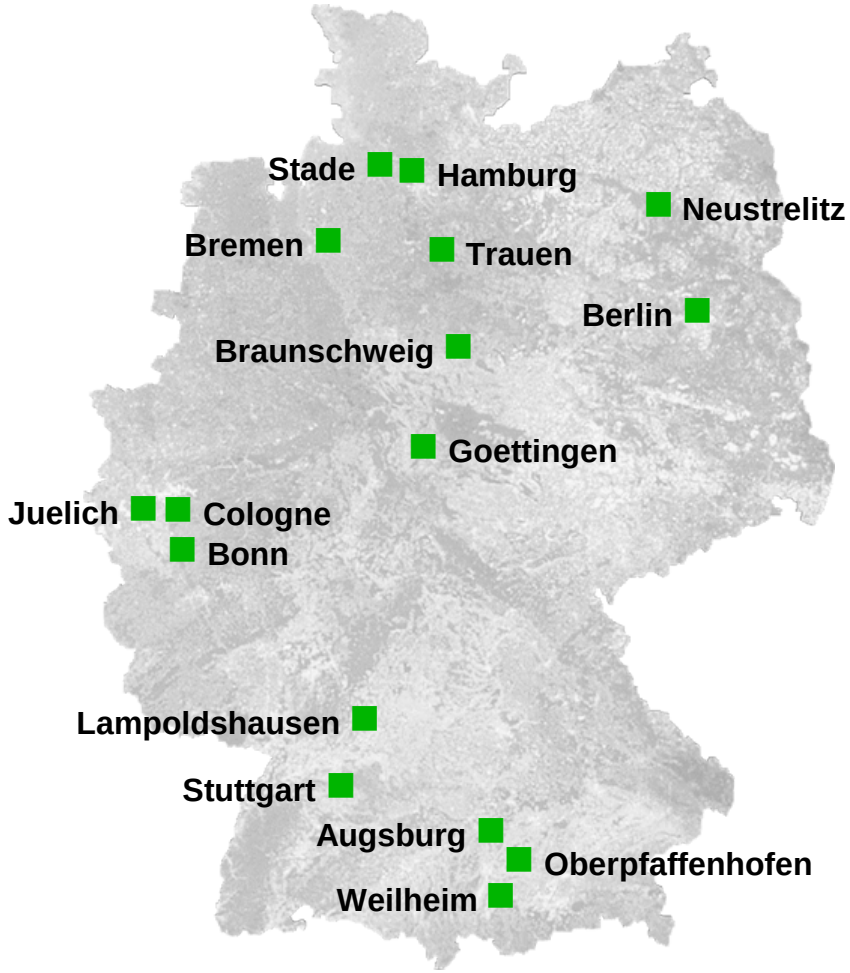


- Research Institution
- Space Agency
- Project Management Agency

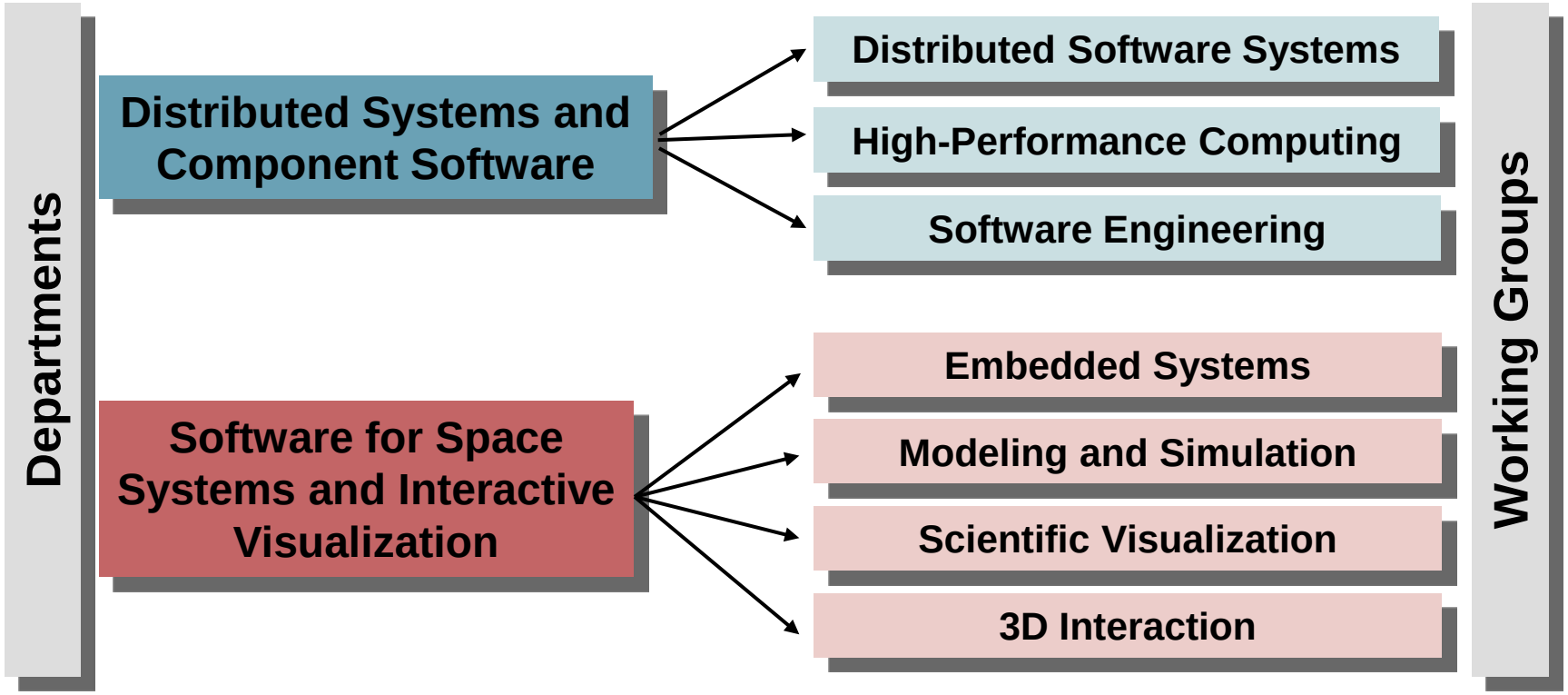
DLR Locations and Employees

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

- Motivation for extreme scale computing
- The DFG project ESSEX
- The ESSEX software infrastructure
- The iterative solver library PHIST
- Iterative solvers from PHIST
  - Methods
  - Performance
- Conclusions

Hypothetical Exascale System

Characteristic	
Flops – peak (PF)	997
Microprocessors	223,872
Cores/microprocessor	742
Cache (TB)	37.2
DRAM (PB)	3.58
Total power (MW)	67.7
Memory bandwidth / Flops	0.0025
Network bandwidth / Flops	0.0008

"Aggressive Strawman" (2007)
DARPA (The Defense Advanced Research Projects Agency of the U.S)

170 million cores!

Today´s Workstations are Hundredfold Parallel

- Example: Intel® Haswell architecture
  - 1-2 CPU sockets
  - with 18 cores each
  - Hyperthreading, 2 threads/core
  - 8 operations performed concurrently (SIMD, FMA)
- GPUs offer parallelism with ten thousands of asynchronous threads.

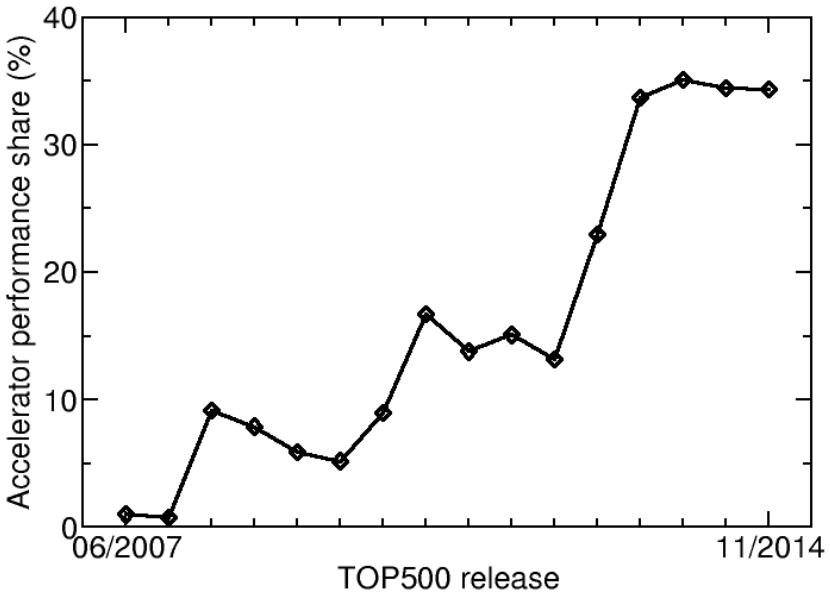
Conclusion: Highly scalable software is not only relevant for high-end computing, but has many applications on common hardware available for everyone.

Accelerator Hardware makes HPC Main Stream

- High parallelism and flop rates
- Expert know-how for porting necessary (e.g. CUDA knowledge)
- higher memory bandwidth
- new bottleneck CPU→device

- Common representatives:

TOP500 development



Nvidia® GPUs

Intel® Xeon Phi

Software Challenges

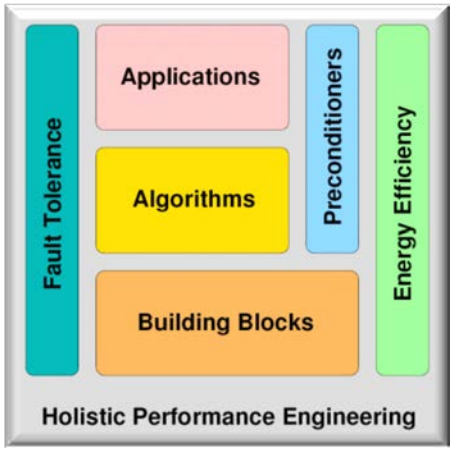
Problems:

- Only a few algorithms are designed for extreme parallelism.
- Applications software is as a rule incrementally adapted to new technologies.

Extreme parallelism requires:

- Extremely scalable algorithms
- New concepts for
  - fault tolerance
  - programming models
  - frameworks for modelling and simulation
- Focus on suitable software engineering methods for parallel codes
  - New test methods
  - New tools for development and analysis

The DFG Project ESSEX



DFG programme

Software for Exascale Computing

SPPEXA

Project ESSEX

Equipping Sparse Solvers for the Exascale

Period: 2013-2015

Extended to 2018

Participating universities

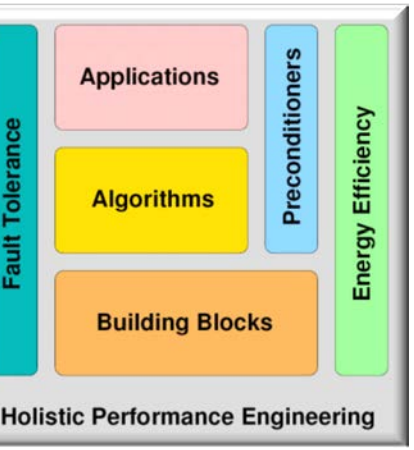
RRZE Erlangen, Computer Science (Prof. Wellein, Hager)
Wuppertal, Numerical Analysis (Prof. Lang)
Greifswald, Physics (Prof. Fehske)

International contacts

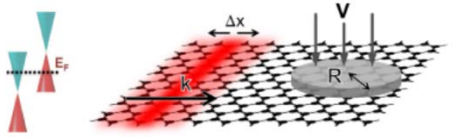
Sandia (Trilinos project)
Tenessee (Dongarra)
Japan: Tsukuba, Tokyo
The Netherlands: Groningen, Utrecht

ESSEX develops open-source software.

ESSEX Motivation



Quantum physics/information applications



Large, Sparse

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

and beyond….

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

"Few" (1,…,100s) of eigenpairs

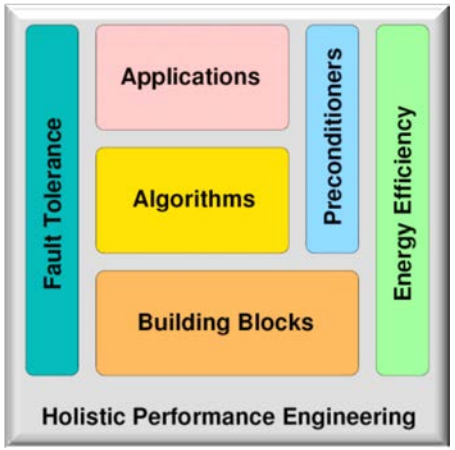
"Bulk" (100s,…,1000s) eigenpairs

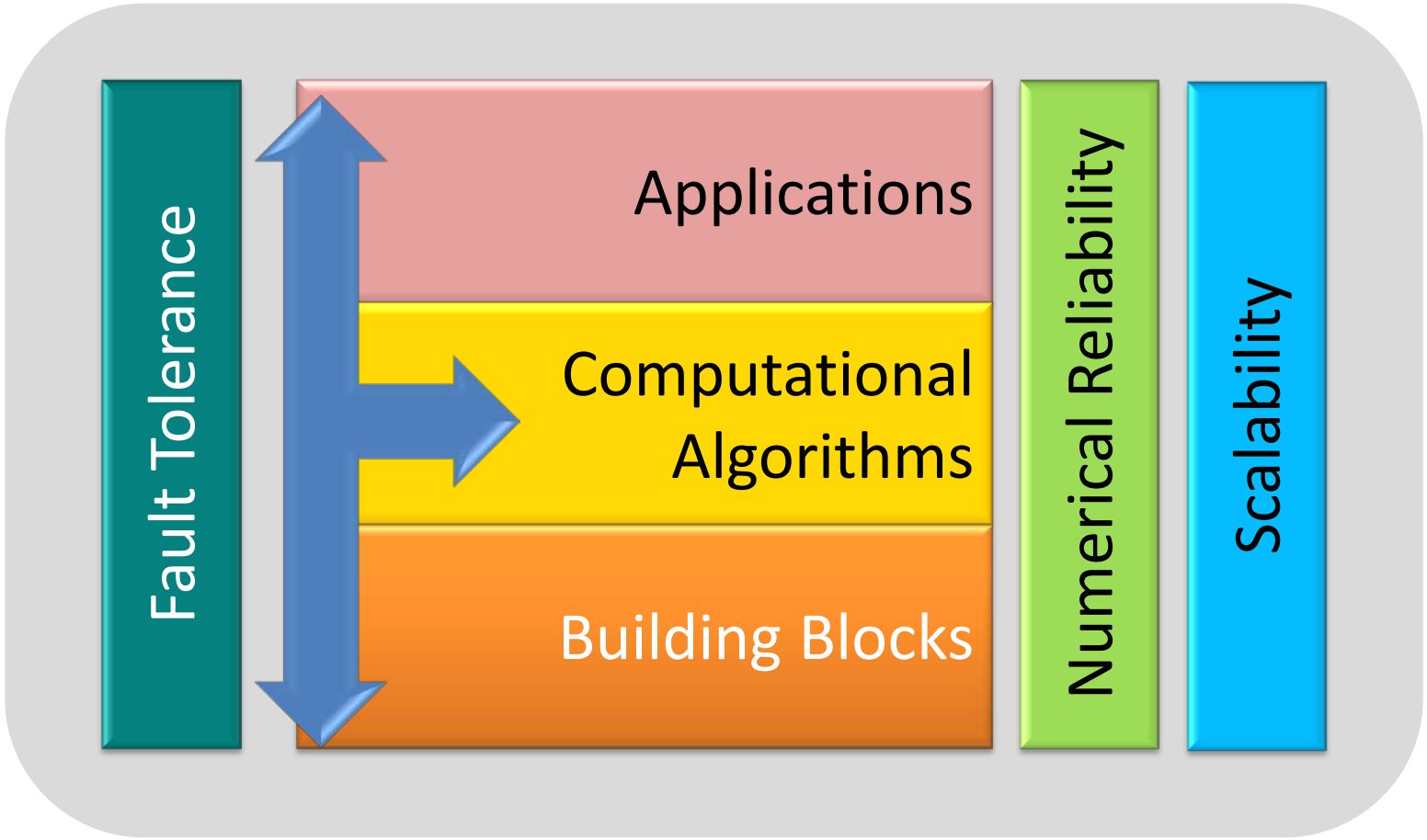
$$\{\lambda_1, \lambda_2, \dots, \dots, \dots, \dots, \lambda_k, \dots, \dots, \dots, \dots, \lambda_{n-1}, \lambda_n\}$$

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

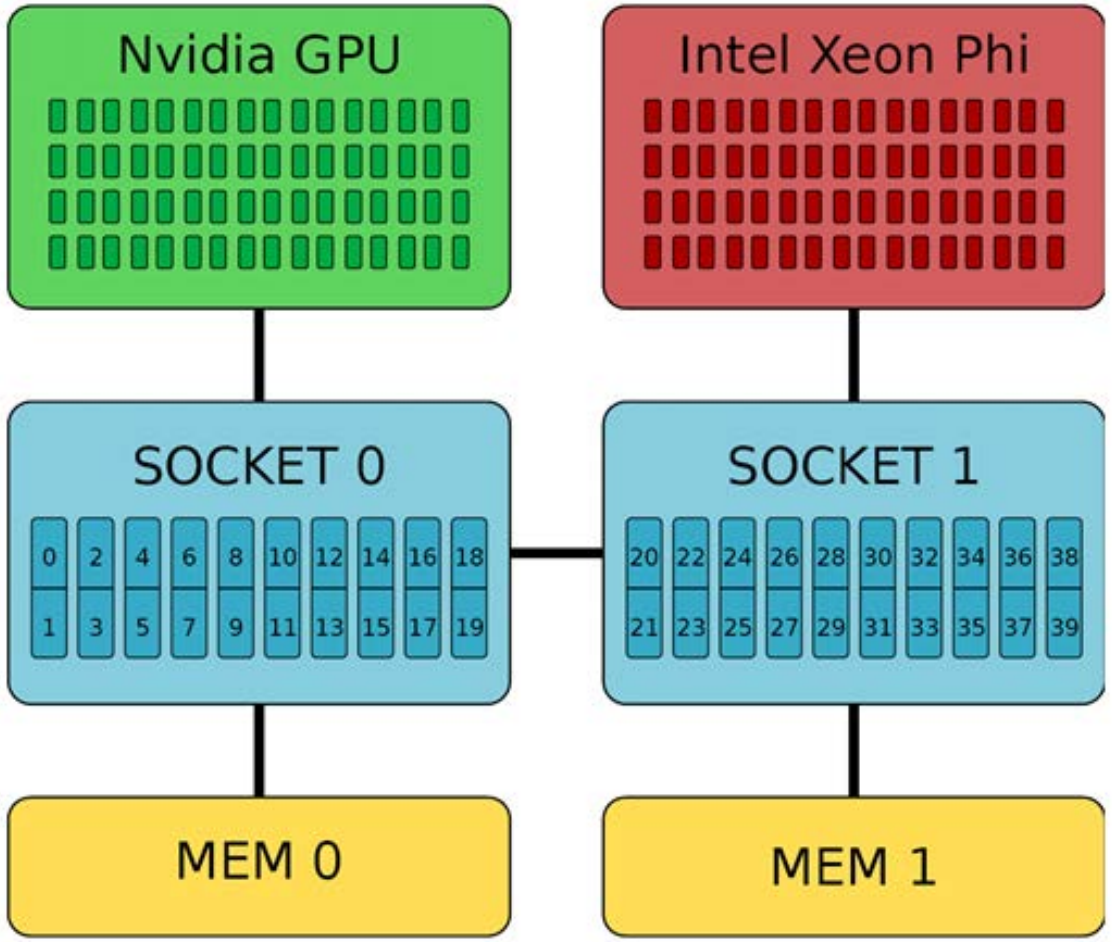
→ Sparse eigenvalue solvers of broad applicability

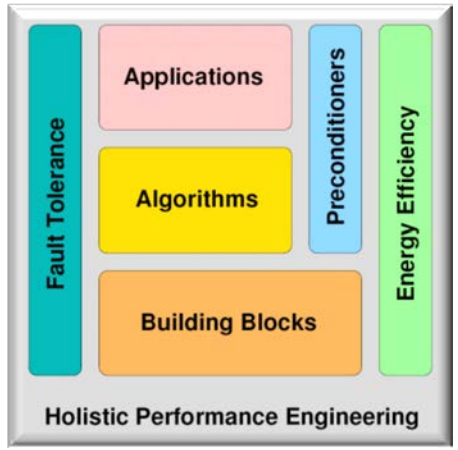
Enabling Extreme Parallelism through Software Codesign





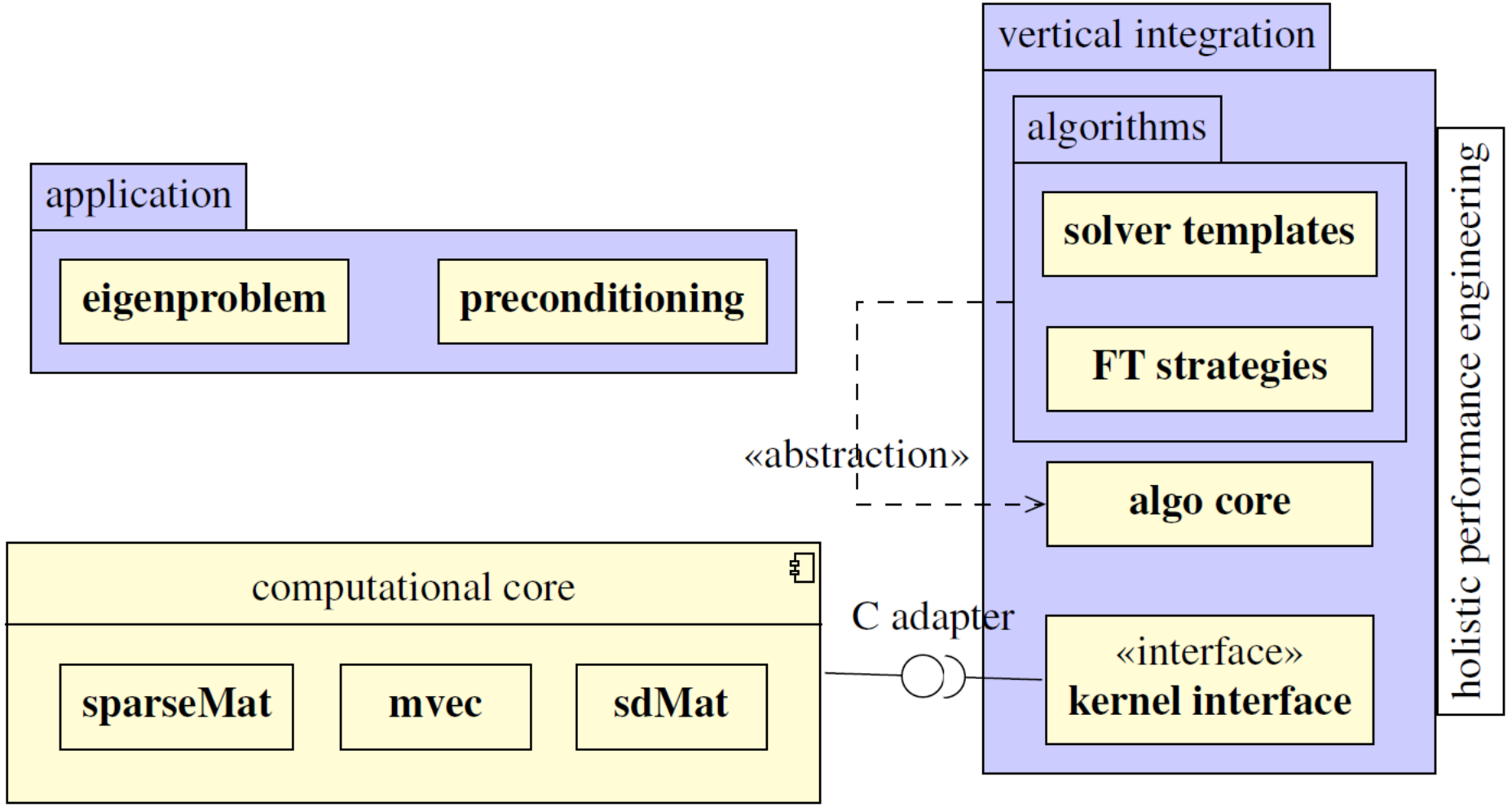
Programming Models for Heterogeneous HPC Systems



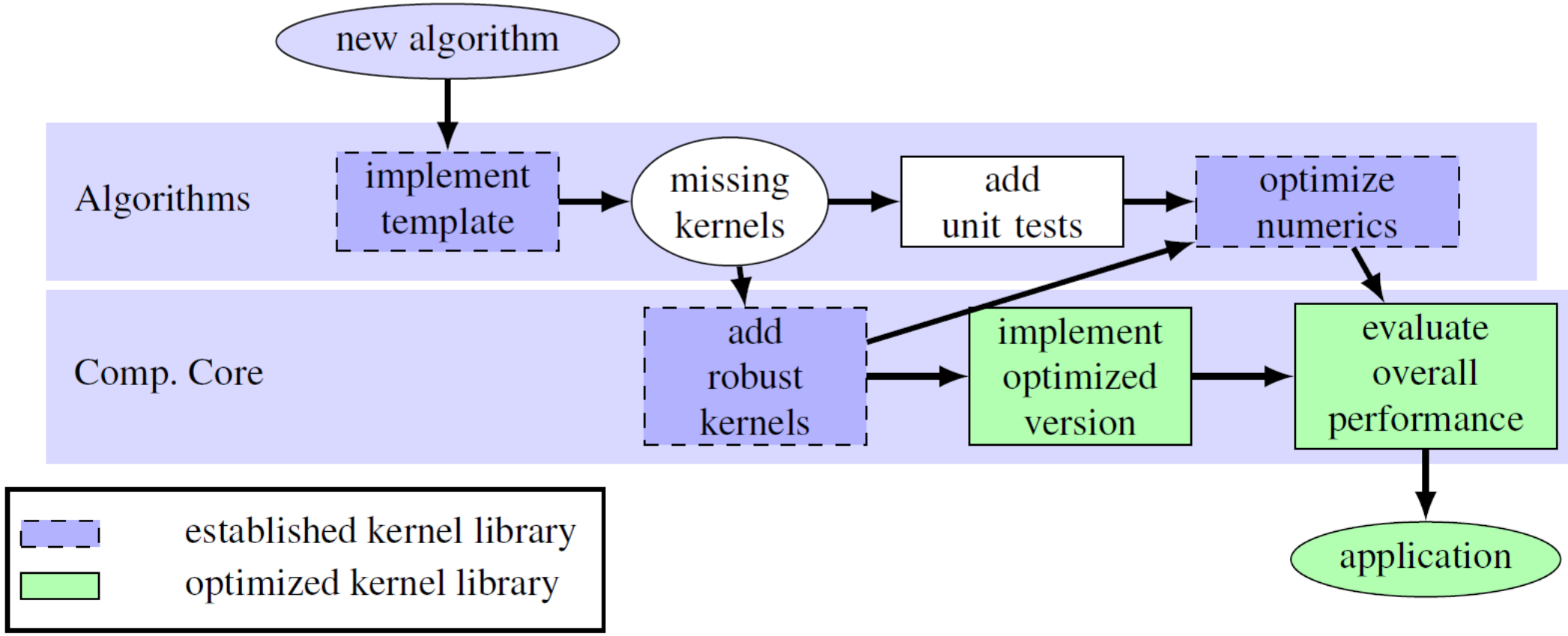


- Flat MPI + off-loading
- Runtime (e.g. MAGMA, OmpSs)
  - Dynamic scheduling of small tasks → good load balancing
- Kokkos (Trilinos)
  - High level of abstraction (C++11)
- **MPI+X strategy in ESSEX**
  - **X: OpenMP, CUDA, SIMD Intrinsics, e.g. AVX**
  - **Tasking for bigger asynchronous functions → functional parallelism**
  - **Experts implement the kernels required.**

The ESSEX Software Infrastructure



The ESSEX Software Infrastructure: Test-Driven Algorithm Development



Optimized ESSEX Kernel Library

General, Hybrid, and Optimized Sparse Toolkit

- MPI + OpenMP + SIMD + CUDA
- Sparse matrix-(block-)vector multiplication
- Dense block-vector operations
- Task-queue for functional parallelism
- Asynchronous checkpoint-restart

Status: beta version, suitable for experienced HPC C programmers

http://bitbucket.org/essex/ghost

BSD License

The Iterative Solver Library PHIST

PHIST **Pipelined Hybrid parallel Iterative Solver Toolkit**

- Iterative solvers for sparse matrices
  - Eigenproblems: Jacobi-Davidson, FEAST
  - Systems of linear equations: GMRES, MINRES, CARP-CG
- Provides some abstraction from data layout, process management, tasking etc.
- Adapts algorithms to use block operations
- Implements asynchronous and fault-tolerant solvers
- Simple functional interface (C, Fortran, Python)
- Systematically tests kernel libraries for correctness and performance
- Various possibilities for integration into applications

Status: beta version with extensive test framework
http://bitbucket.org/essex/phist
BSD License

Solvers & components

block JDQR, CARP-CG
Krylov (e.g. GMRES)
(block) orthogonalization

Tests
Integration
Unit

Kernel interface
- simple C function interface
- does not prescribe data layout
-
examples:

$$Y = \alpha AX + \beta Y$$
$$X = DKSWP(A - \sigma I, X, B, \omega)$$
$$C = V^T Y, Y = Y - V \cdot C$$

Integration of PHIST into Applications

Selection of kernel library

Required flexibility

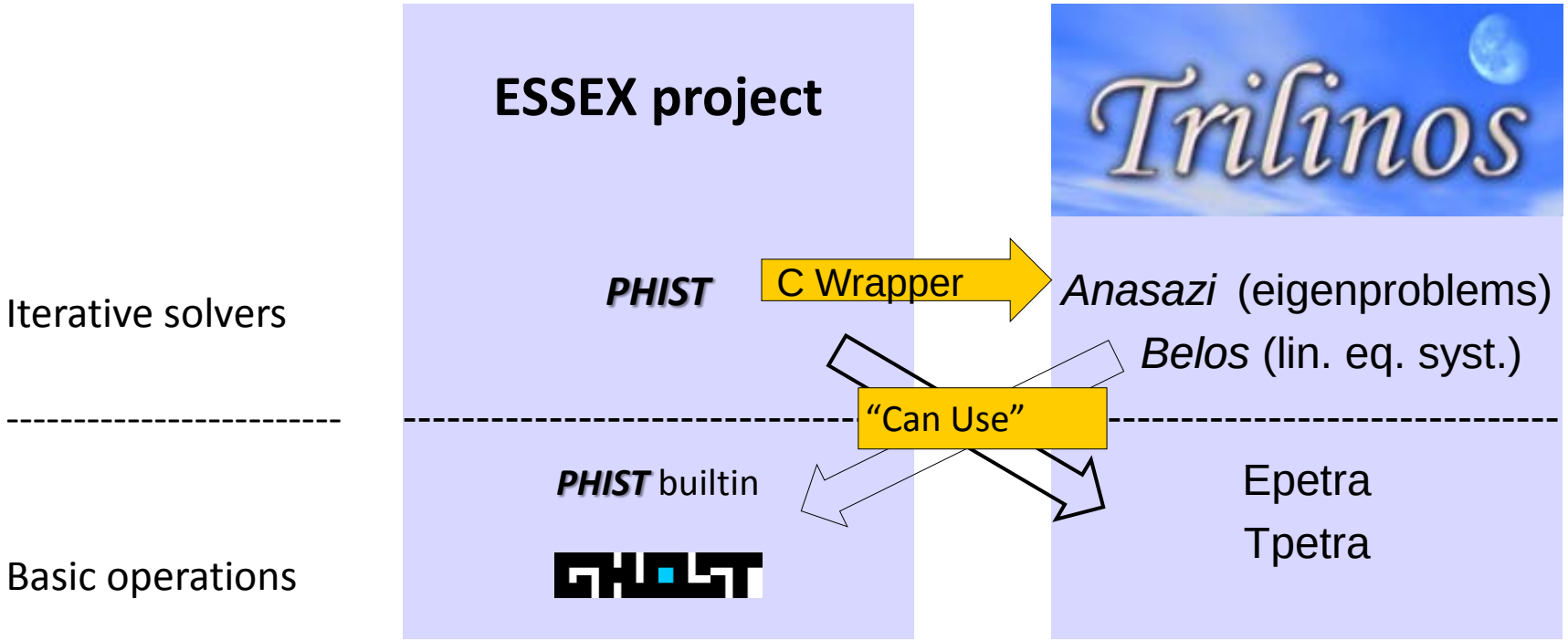
low **high**

No easy access to matrix elements	***PHIST*** builtin Only CPU F‘03+OpenMP CRS format	Trilinos Various arch. Large C++ code base	Own data structures Adapter ca 1000 lines of code

Hardware awareness

high **low**

Interoperability of PHIST and Trilinos



Iterative Solvers from PHIST: Jacobi-Davidson QR method (Fokkema, 1998)

Sketch of the algorithm

1: **while** not converged **do** ▷ Outer iteration
2: Project the problem to a small subspace
3: Solve the small eigenvalue problem
4: Calculate an approximation and its residual
5: Approximately solve the correction equation ▷ Inner iteration
6: Orthogonalize the new direction
7: Enlarge the subspace
8: **end while**

Iterative Solvers from PHIST: Block JDQR method

Idea

- Calculate corrections for n_b eigenvalues at once
- Block correction equation with $\tilde{Q} = \begin{pmatrix} Q & \tilde{v}_1 & \dots & \tilde{v}_{n_b} \end{pmatrix}$:

$$(I - \tilde{Q}\tilde{Q}^T)(A - \tilde{\lambda}_i I)(I - \tilde{Q}\tilde{Q}^T)w_{k+i} = -r_i \qquad i = 1, \dots, n_b$$

$\rightarrow$ Approximately solve n_b linear systems at once

- Provides new directions $w_{k+1}, \dots, w_{k+n_b}$ for the subspace iteration

Numerical properties

- More robust
- Usually needs more operations

Iterative Solvers from PHIST: Complete Block JDQR method

Sketch of the complete algorithm

1: Setup initial subspace
2: **while** not converged **do**
3: Project the problem to a small subspace
4: Solve the small eigenvalue problem
5: Calculate an n_b approximations and their residual
6: Lock converged eigenvalues
7: Shrink subspace if required (thick restart)
8: Approximately solve the n_b correction equations
9: Block-Orthogonalize the new directions (using TSQR)
10: Enlarge the subspace
11: **end while**

Block JDQR method: Required Linear Algebra Operations

Sparse matrix-multiple-vector multiplication (spMMVM)

- Large distributed sparse matrix A in SELL-C-σ format
- Distributed blocks of vectors $X, Y \in \mathbb{R}^{n \times n_b}$
- Shifted spMMVM: $y_i \leftarrow (A - \tilde{\lambda}_i I)x_i, \quad i = 1, \ldots, n_b$

Block vector operations

- Different types of operations:

	local	all-reduction
BLAS 1	$Y \leftarrow X + Y$	$\|x_i\|, i = 1, \ldots, n_b$
BLAS 3	$Y \leftarrow XM$	$M \leftarrow X^T Y$

- Redundantly stored small matrices $M \in \mathbb{R}^{n_b \times n_b}$

Block JDQR method: Block Vector Operations

Background

- All operations are memory bound.
  (also the GEMM, as all matrices are very tall and skinny)

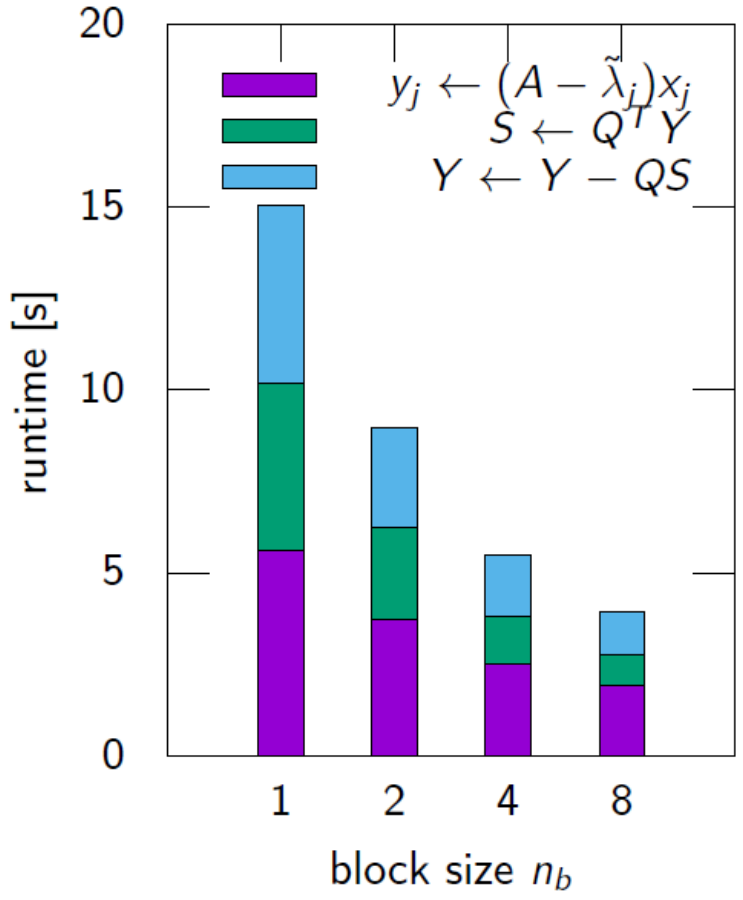
Results of blocking

- Faster BLAS 3 operations (e.g. $Y \leftarrow XM$)
- Message aggregation for all-reductions (e.g. $M \leftarrow X^T Y$)

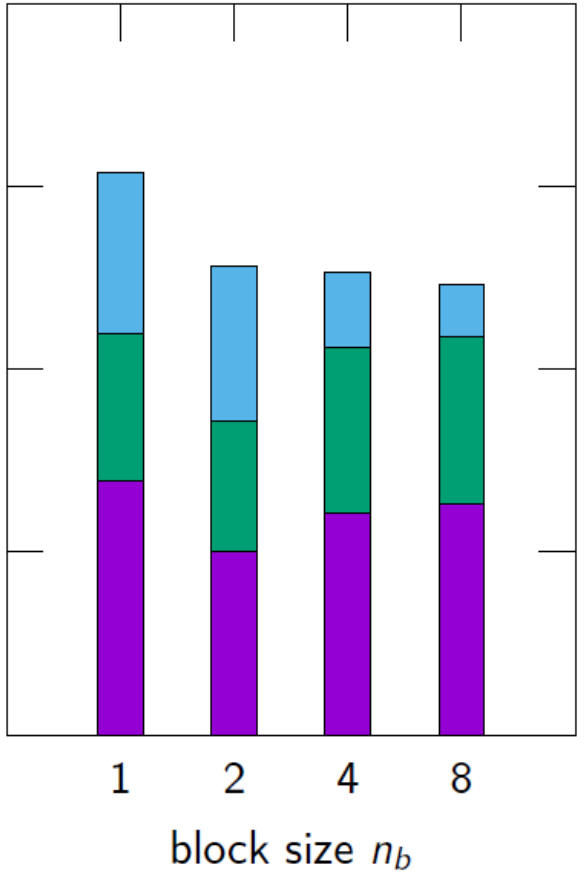
$\rightarrow$ Improved performance of some operations

Block JDQR method: Result of Correction Kernel Optimization

10-core Intel Ivy Bridge CPU; CRS format; matrix: 10^7 rows; $1.5 \cdot 10^8$ nonzeros; 120 correction operations



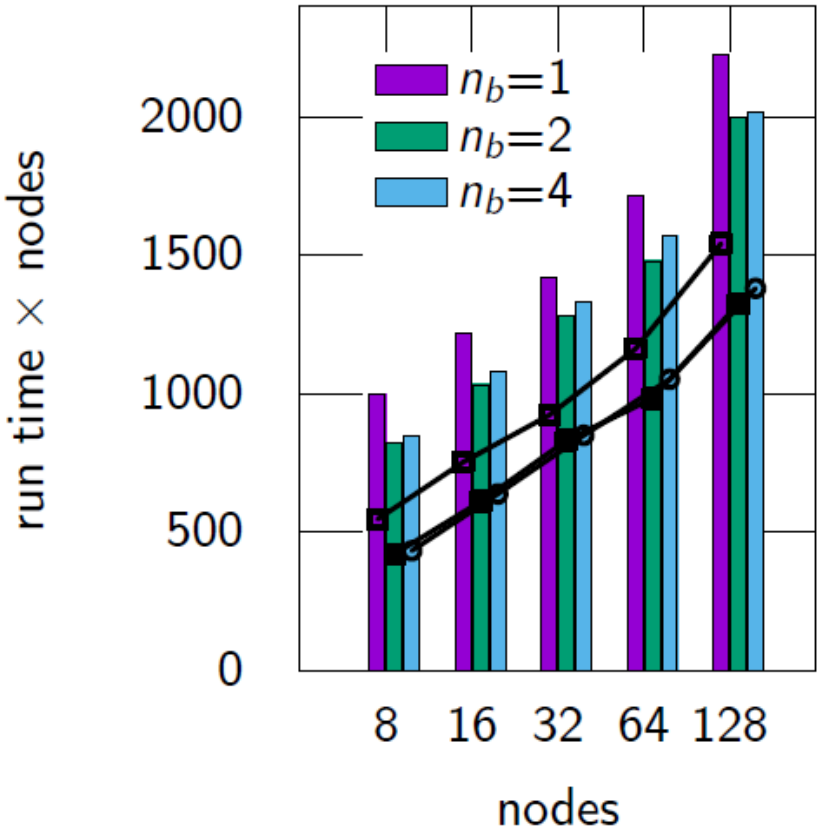
GHOST, row-major blockvectors



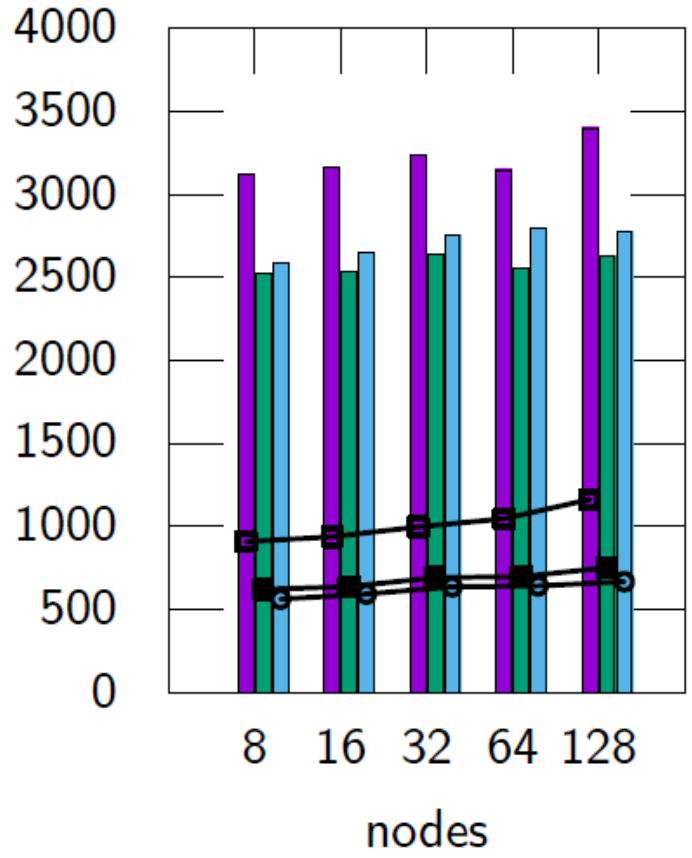
Tpetra, col.-major blockvectors

Block JDQR method: Overall Speedup through Blocking

Node: 2x10-core Intel Ivy Bridge CPU; SELL-C-σ format; blocked preconditioning; residual reduction: 10^{-8}

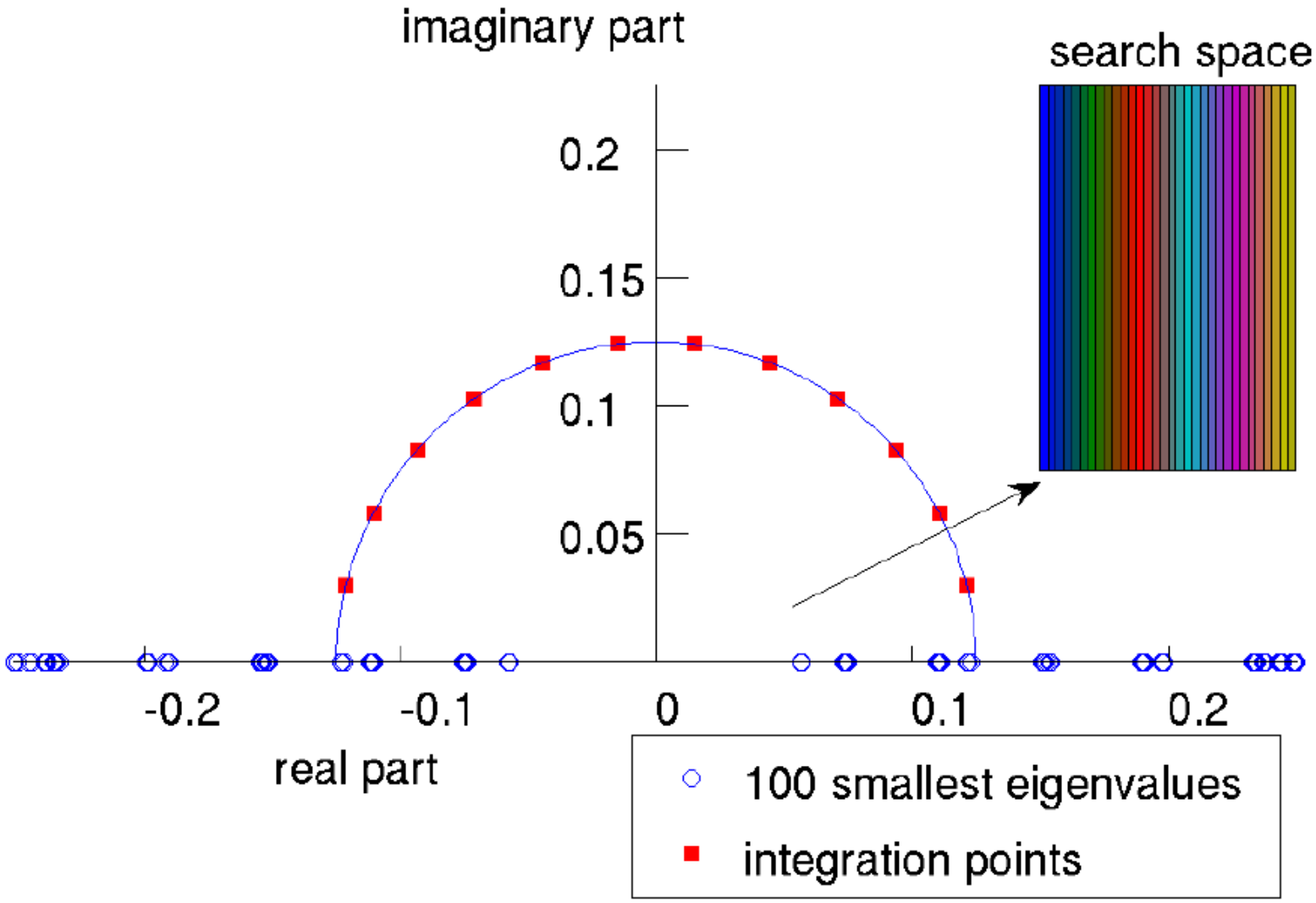


20 left-most eigenpairs of $Spin_{SZ}[28]$
$n \approx 40M$, MINRES as 'preconditioner'



20 largest eigenpairs, 3D conv-diff.
$n = 512^3 \approx 134M$, GMRES

Iterative Solvers from PHIST: FEAST eigensolver (Polizzi '09)



Iterative Solvers from PHIST: Linear Systems for FEAST/graphene

Tough:

- very large ($N \sim 10^9$ carbon atoms)
- complex symmetric and completely indefinite
- small random numbers on and around the diagonal
- spectrum essentially continuous
- shifts get very close to the spectrum

But also nice in some ways:

- 2D mesh, very sparse ($\sim$ 10 entries/row)
- many RHS/shift (block methods, etc.)

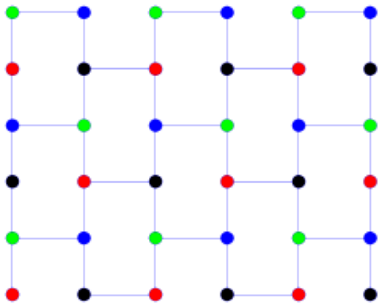
State of the art: direct solver, feasible up to a few million C atoms

Iterative Solvers from PHIST: The CGMN Linear Solver

- Björck and Elfving, 1979
- CG on the semi-definite problem $(I - Q_{SSOR})x = b$, where $Q_{SSOR} = Q_1 Q_2 \ldots Q_N Q_{N-1} \ldots Q_1$ is the SSOR iteration on $AA^T y = b$
- $Q_i v = (I - \omega \frac{a_i^T v}{a_i a_i^T}) a_i^T$: project v onto a_i (row i of A)
- extremely robust: A may be singular, non-square etc.
- squaring A remedies small diagonal entries
- row scaling alleviates issue of 'squared condition number'

Iterative Solvers from PHIST: Parallelization Strategies for CGMN

Algebraic Multi-Coloring



Distance-2 coloring resolves data dependency

- yields fine grained parallelism (e.g. GPGPU)

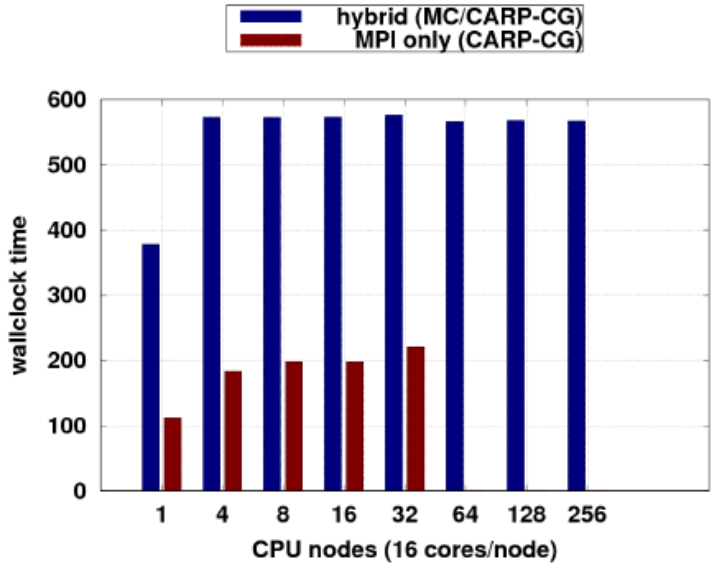
CARP: Component-Averaged Row Projection (Gordon & Gordon, 2005)

- sequential sweeps on subdomains
- exchange and average halo elements
- retains convergence properties of sequential algorithm

Idea: node-local MC with MPI-based CARP between the nodes

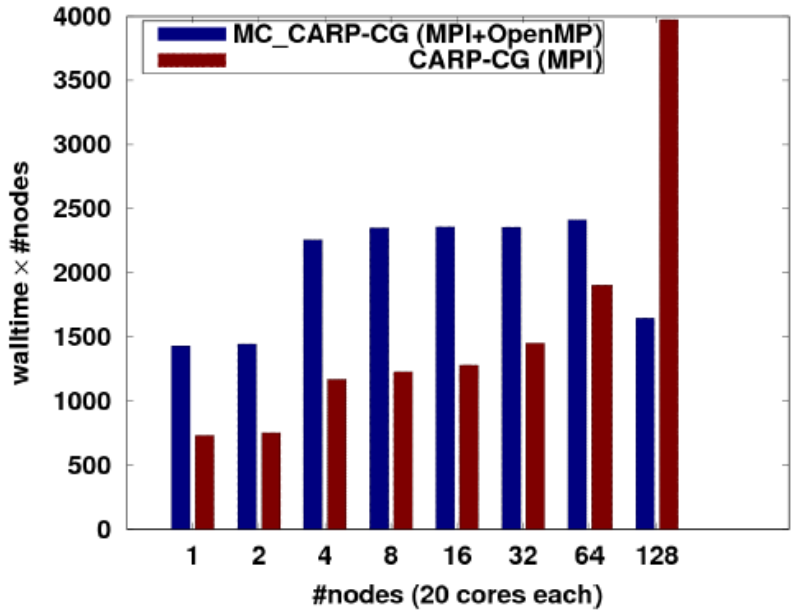
Iterative Solvers from PHIST: Scaling of CARP-CG

Intel Ivy Bridge



Graphene, 20M atoms/node (up to 5B)

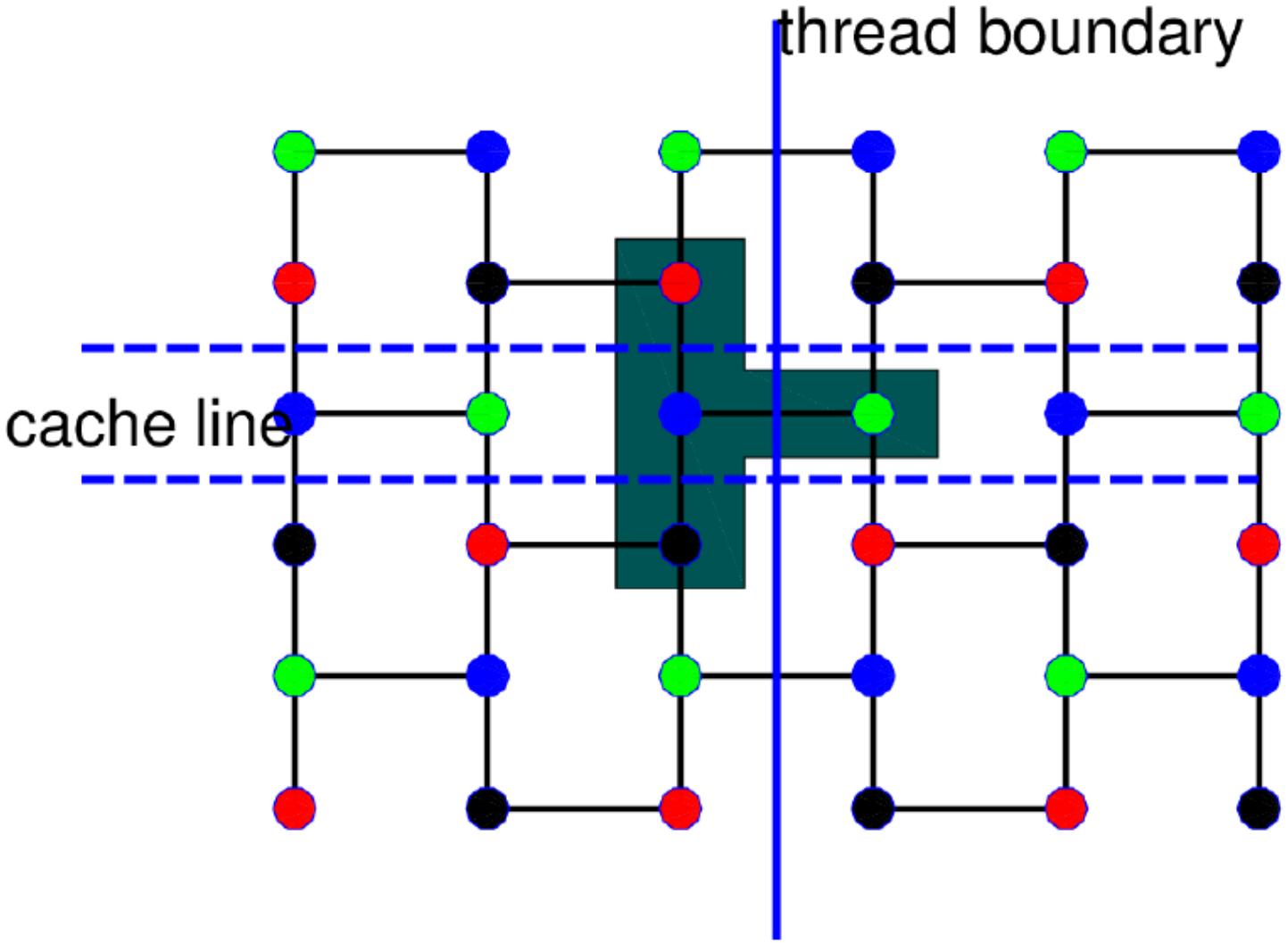
- Coloring costs performance;
- but is more memory efficient;



Strong scaling, 20M unknowns in total

- yields better strong scaling;
- and has better potential for GPUs and Xeon Phi.

MC-CARP-CG: Cache Coherence Kills Performance on Socket Level



Conclusions

- ***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 JDQR.
  - Slight increase of operations
  - Impact of memory layout: row- rather than column-major for block vectors
  - Higher node-level performance
  - Inter-node advantage: message aggregation
- CGMN with CARP and 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
- **Future:** AMG preconditioning for blocked JDQR & FEAST (Kengo Nakajima, University of Tokyo); exploitation of the non-linear Sakurai-Sugiura method (Tetsuya Sakurai, University of Tsukuba)

References

- Röhrig-Zöllner, Thies, Basermann et al.: ***Increasing the performance of Jacobi-Davidson by blocking***; SISC (in print)
- Thies, Basermann, Lang et al.: ***On the parallel iterative solution of linear systems arising in the FEAST algorithm for computing inner eigenvalues***; Parallel Computing 49 (2015) 153–163

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

DLR

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