Dünnbesetzte Eigenwertlöser auf Heterogenen Supercomputern

Dr. Jonas Thies

DLR Simulations- und Softwaretechnik
Abteilung Verteilte Systeme und Komponentensoftware
Arbeitsgruppe High Performance Computing

DLR

Wissenschaftliche Einbettung

DFG Schwerpunktprogramm

Software for Exascale Computing

Projekt ESSEX

Equipping Sparse Solvers for the Exascale

Laufzeit: 2013-2015

Folgeantrag gestellt

Beteiligte Universitäten

RRZE Erlangen (Prof. Wellein, Hager)
Wuppertal, Numerik (Prof. Lang)
Greifswald, Physik (Prof. Fehske)

Internationale Kontakte

Sandia (Trilinos Projekt)
Tenessee (Dongarra)
Japan: Tsukuba, Tokyo
Niederlande: Groningen, Utrecht

Software wird unter einer OpenSource Lizenz zur Verfügung gestellt

Übersicht

- **Aktuelle Herausforderungen für High-End HPC**
- Eigenwertprobleme: Anwendungen und Algorithmen
- Technologie: MPI+X, Fehlertoleranz und Nodelevel Performance
- Software Entwicklungen: GHOST und PHIST
- Fallbeispiele und Zusammenfassung

Hypothetisches 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 Millionen Kerne!

Schon heutige Workstations sind 100-fach parallel

- Beispiel: Intel® Haswell Architektur

  1-2 CPU Sockel (Sockets)
  je 18 Kerne (Cores)
  Hyperthreading, 2 Threads/Core
  Je 8 Operationen gleichzeitig (SIMD, FMA)

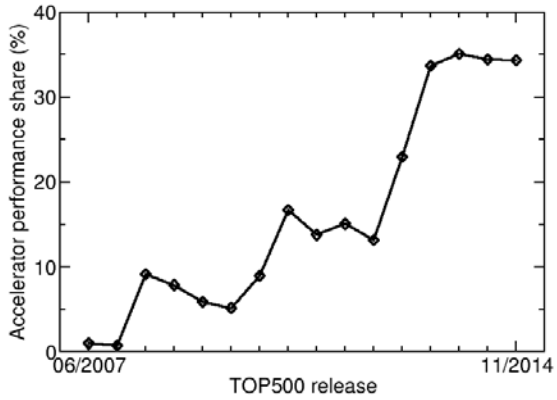
- Grafikkarte (tausende Threads) oft verfügbar aber selten genutzt

FAZIT: Die hier vorgestellte Software ist nicht nur für das High-End Computing relevant sondern kann vielfältig eingesetzt werden

Beschleunigerhardware wird zum „HPC Main Stream“

- Hohe Parallelität und Flop Raten
- Experten für die Portierung erforderlich (z.B. CUDA Kenntnisse)
- höhere Speicherbandbreite
- neuer Bottleneck CPU→Device

TOP500 Entwicklung



Die häufigsten Vertreter:

Nvidia® GPUs

Intel® Xeon Phi

Software-Herausforderungen

Probleme

- Nur wenige Algorithmen sind für extreme Parallelität geeignet.
- Bestehende Anwendungssoftware wurde für moderat parallele Systeme entwickelt (üblicherweise „MPI flat“)

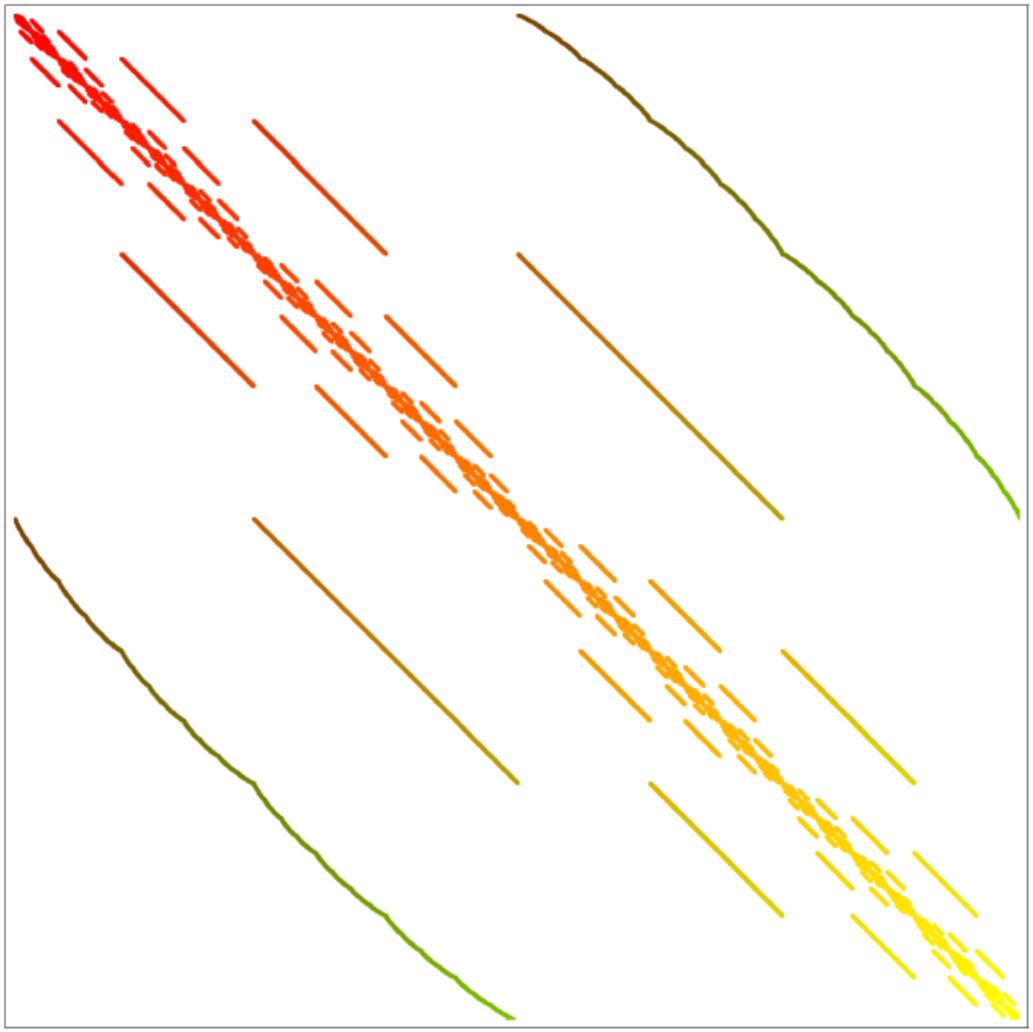
Peta- bis Exascale erfordert

- Extrem skalierbare Algorithmen
- Neue Konzepte für
  - Ausfallsicherheit (Fault Tolerance)
  - Programmiermodelle
  - Software Engineering (Methoden und Tools)

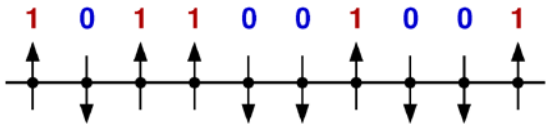
Übersicht

- Aktuelle Herausforderungen für High-End HPC
- **Eigenwertprobleme : Anwendungen und Algorithmen**
- Technologie: MPI+X, Fehlertoleranz und Nodelevel Performance
- Software Entwicklungen: GHOST und PHIST
- Fallbeispiele und Zusammenfassung

Dünnbesetzte Matrizen



Beispiel Quantenmechanik



Spin-Kette von N Elektronen (2^N mögliche Zustände)

- Dünnbesetzte Matrizen haben

„so wenige nicht-Null Einträge, daß es sich lohnt, diese Eigenschaft auszunutzen“

Lineare Eigenwertprobleme

- Eine Matrix **A** beschreibt einen Zustandsübergang
- ihre Eigenwerte **λ** und Eigenvektoren x beschreiben die dynamischen Eigenschaften eines Systems:

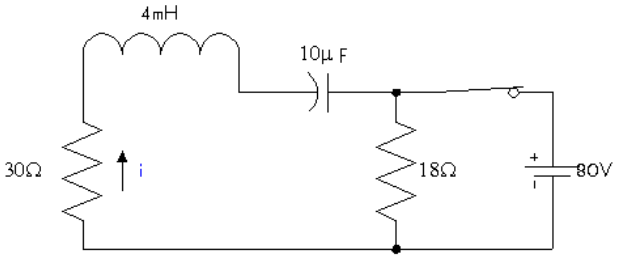
$$\mathbf{A}x = \lambda x$$

Eigenwerte und –vektoren sind wertvoll in der Praxis, z.B. Energieniveaus von Quantensystemen:

$$\mathbf{H}\psi = E\,\psi$$ (Schrödinger Gleichung)

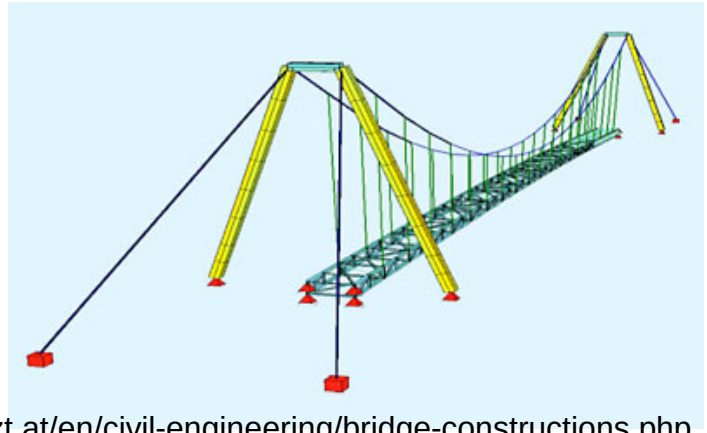
Anwendungen dünnbesetzter Eigenwertprobleme

Stromnetze



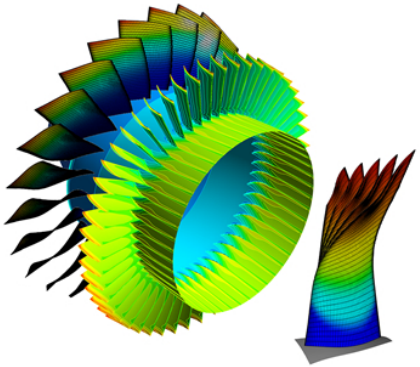
www.mhhe.com/engcs/electrical/hkd/tutorials/Tut9-1.htm

Vibrationen



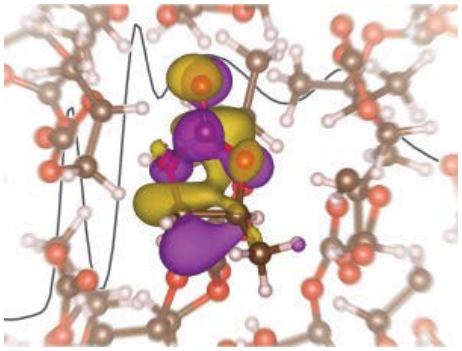
www.ssd-zt.at/en/civil-engineering/bridge-constructions.php

Stabilitäts-analyse (z.B. CFD)



idac.co.uk/products/products/cfx.htm

Phys. Chemie, Elektronen-struktur



newscenter.lbl.gov/2014/12/19/better-electrolyte-for-lithium-ion-batteries/

Wie berechnet man Eigenwerte?

- Eigen-Information wird aus Matrix-Vektor Operationen extrahiert

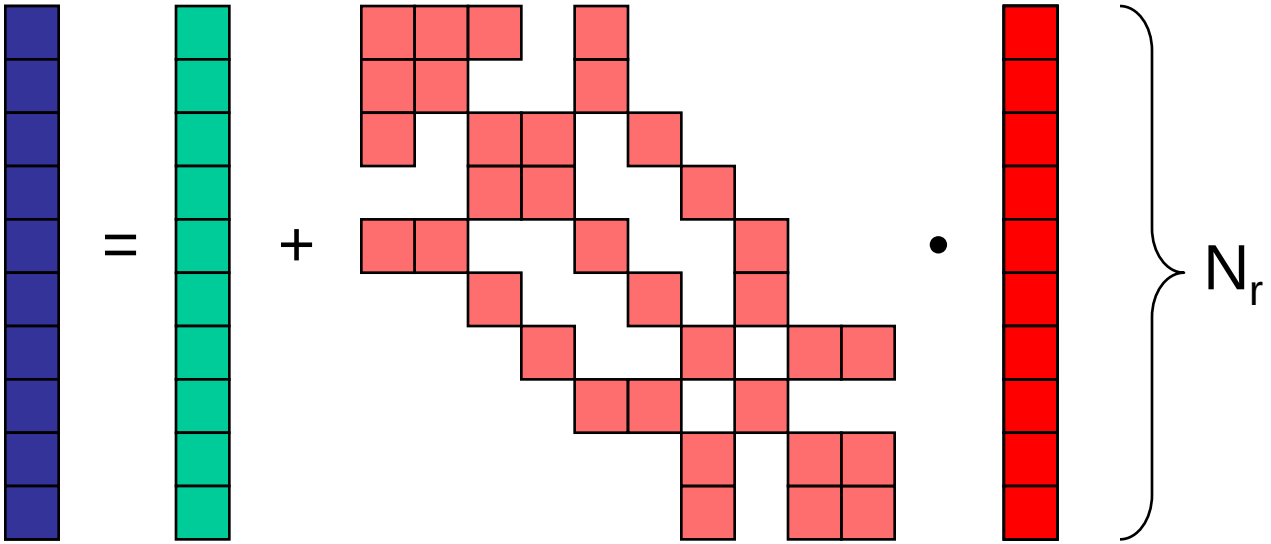
Algorithm 1: Simple Power Method for finding $x, \lambda : \mathrm{A}x \approx \lambda x$

Input: A, tol, start vector y
  for $k = 1, 2, \ldots$ **do**
    $v = y/||y||_2$;
    $y = \mathrm{A} \cdot v$;
    $\theta = v^T y$;
    if $||y - \theta v||_2 <$ tol **then** break;
    end if
  end for
  result: $\lambda = \theta, x = v$

Beispiel: Krylov Methoden konstruieren eine orthogonale Basis für die „Power Vektoren“, z.B. Lanczos Algorithmus

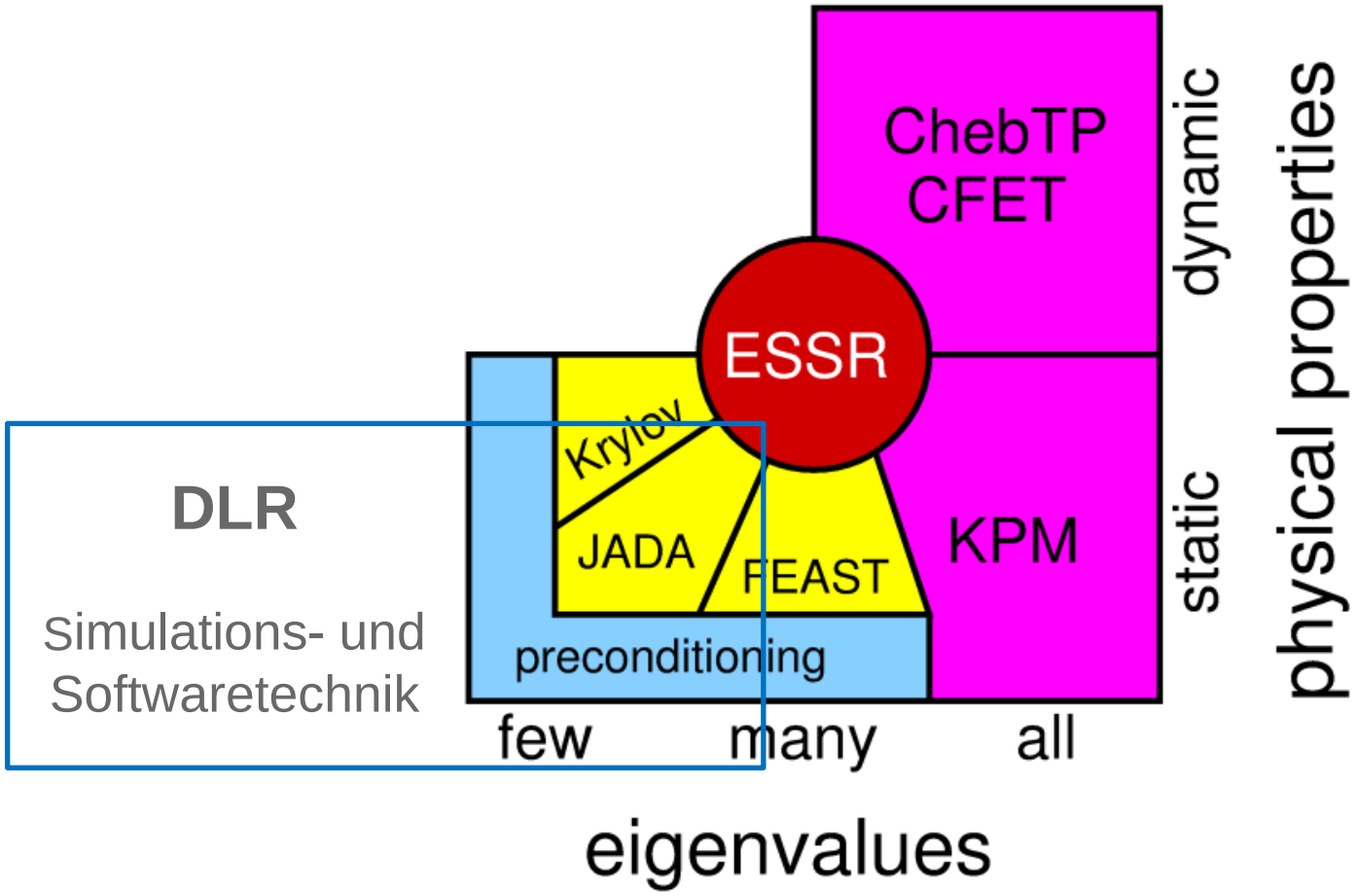
Grundoperation: Dünnbesetztes Matrix-Vektor Produkt

- Dominante Operation in vielen iterativen Methoden
- Speicherbedarf: N_{nz} Elemente der Matrix und Vektoren x,y der Länge N_r
- “Sparse”: $N_{nz} \sim N_r$
- Speichergebundene Operation



Benötigt i.A. indirekte Adressierung bzw. Kommunikation

Algorithmen im ESSEX Projekt

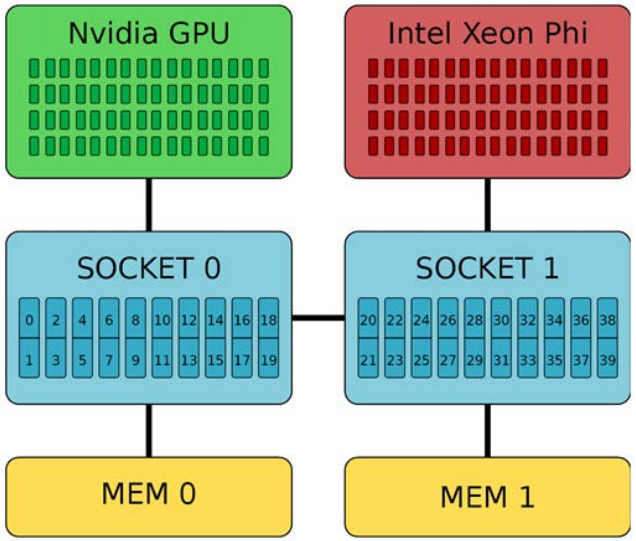


Übersicht

- Aktuelle Herausforderungen für High-End HPC
- Eigenwertprobleme: Anwendungen und Algorithmen
- **Technologie: MPI+X, Fehlertoleranz, Nodelevel Performance**
- Software Entwicklungen: GHOST und PHIST
- Fallbeispiele und Zusammenfassung

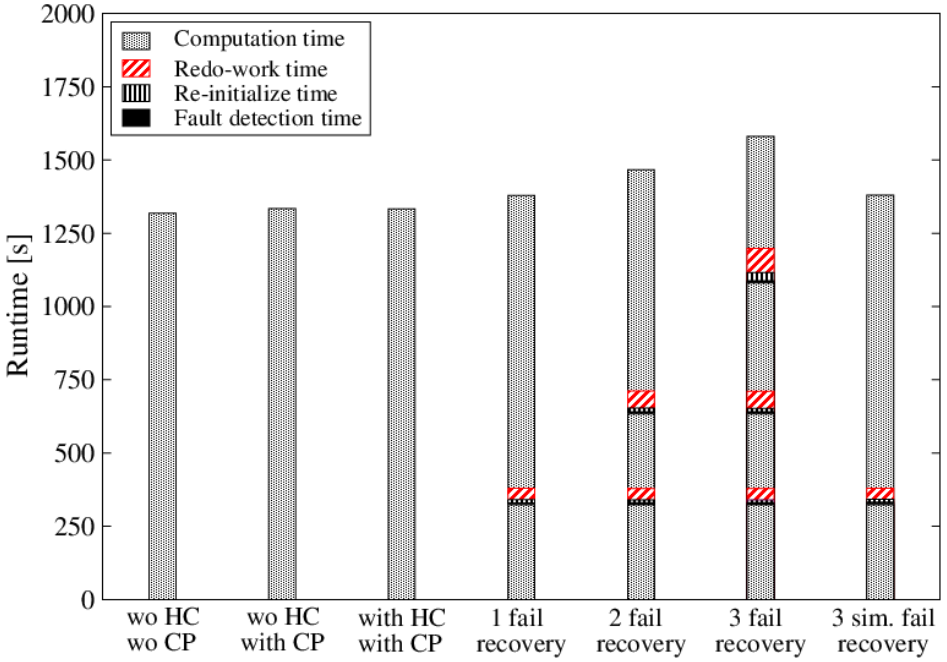
Programmiermodelle für heterogene HPC Systeme

- MPI Flat + Off-loading
- Runtime (z.B. MAGMA, OmpSs)
  - Dynamisches Scheduling von kleinen Tasks (gutes Load Balancing)
- Kokkos (Trilinos)
  - Hoher Grad an Abstraktion (C++11)
- MPI+X Ansatz in GHOST (ESSEX)
  - X: OpenMP, CUDA, SIMD Intrinsics, z.B. AVX
  - Tasking für größere asynchrone Aufgaben (funktionale Parallelität)
  - Experten implementieren benötigte Kernel



Hardware-Defekte überstehen

- Anwendung schreibt **asynchron** „Checkpoints“ **(CP)**
  - auf einer lokalen Platte
  - auf dem Nachbarknoten
- Dedizierter Prozess führt „Health Checks” **(HC)** aller Knoten durch (GASPI/GPI statt MPI)
- Wenn ein Knoten ausfällt:
  - Pool von Ersatzprozessen
  - „Rollback“ zum letzten Checkpoint



Overhead für Recovery ca 18 Sek.
(+ zu wiederholende Berechnungen)

Optimierung 1: Kernel Fusion

- Daten nutzen, während sie im L1 Cache sind
- Beispiel: berechne (mit x,y,z großen Vektoren und A einer Matrix)

$$\mathbf{z} = \mathbf{y} + \alpha \mathbf{A}\mathbf{x},\ c_1 = \left|\left|\mathbf{x}\right|\right|_2,\ c_2 = \left|\left|\mathbf{A}\mathbf{x}\right|\right|_2$$

Mit einzelnen Kerneln (z.B. sparse BLAS)

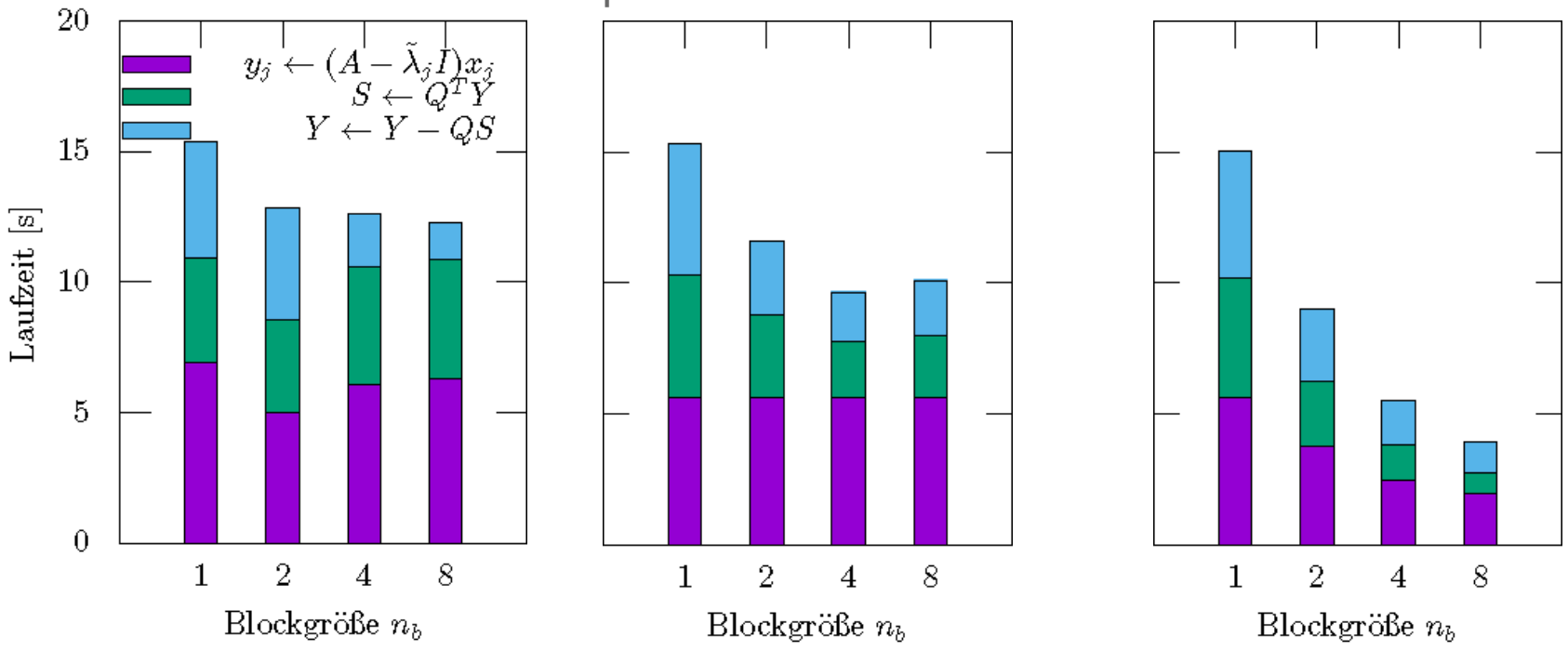
$$\mathbf{t} = \mathbf{A}\mathrm{x},\ \mathrm{z} = \mathrm{y} + \alpha \mathrm{t},\ c_1 = \left|\left|\mathrm{x}\right|\right|_2,\ c_2 = \left|\left|\mathrm{t}\right|\right|_2$$

  - **mindestens** 5 Vektoren werden aus dem Speicher geladen

- In einer einzigen (komplexeren) Funktion:
  - mindestens 2 Vektoren werden geladen

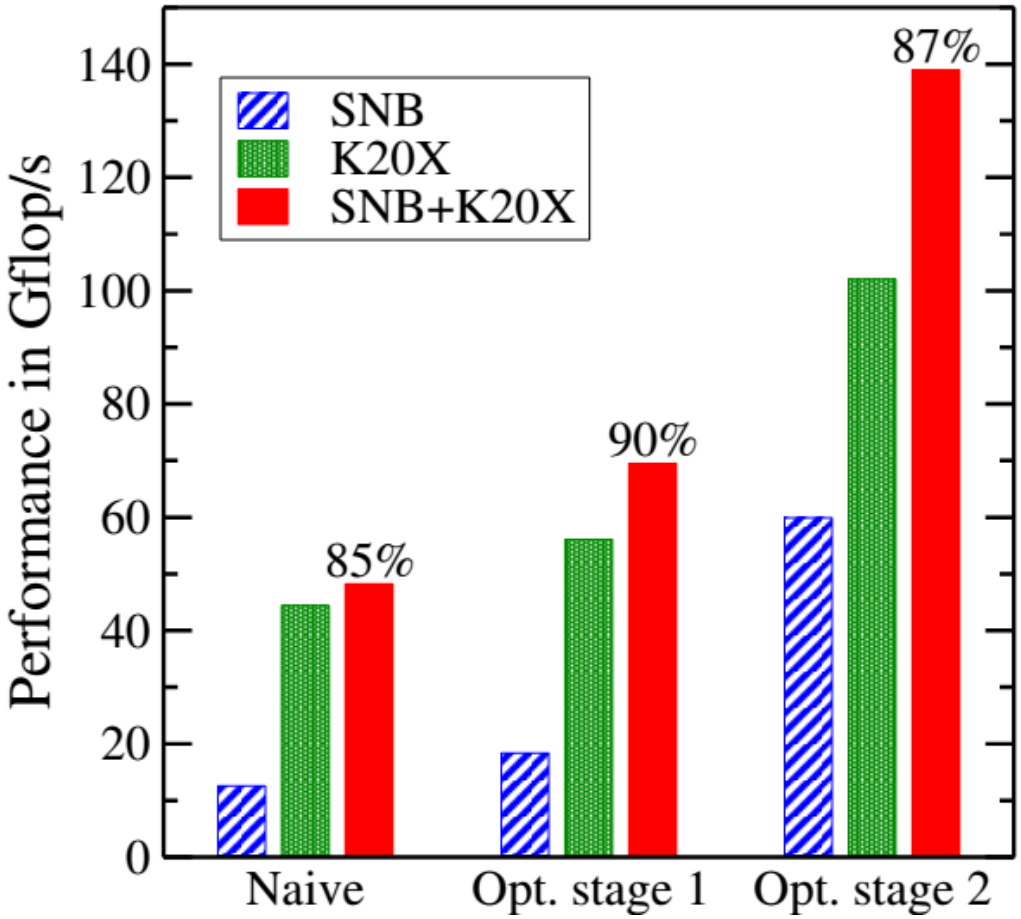
Optimierung 2: Blockvektor Operationen

- Jacobi-Davidson: spMVM gefolgt von Projektion, $(\mathrm{I} - QQ^T)(\mathbf{A} - \sigma\mathrm{I})\mathrm{x}$
- prinzipiell bessere Datenlokalität wenn x mehrere Spalten hat (SpMMVM)
- Weniger Synchronisationspunkte/Nachrichten



Tpetra (Trilinos) GHOST (Naiv) GHOST (row-major)

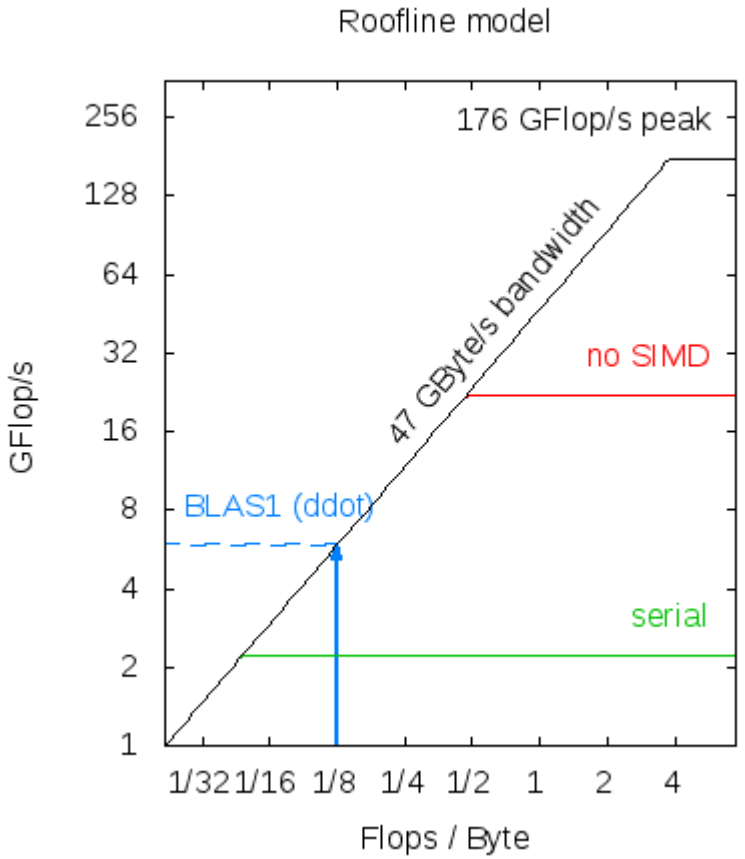
Beispiel: KPM Algorithmus auf Heterogenem Knoten



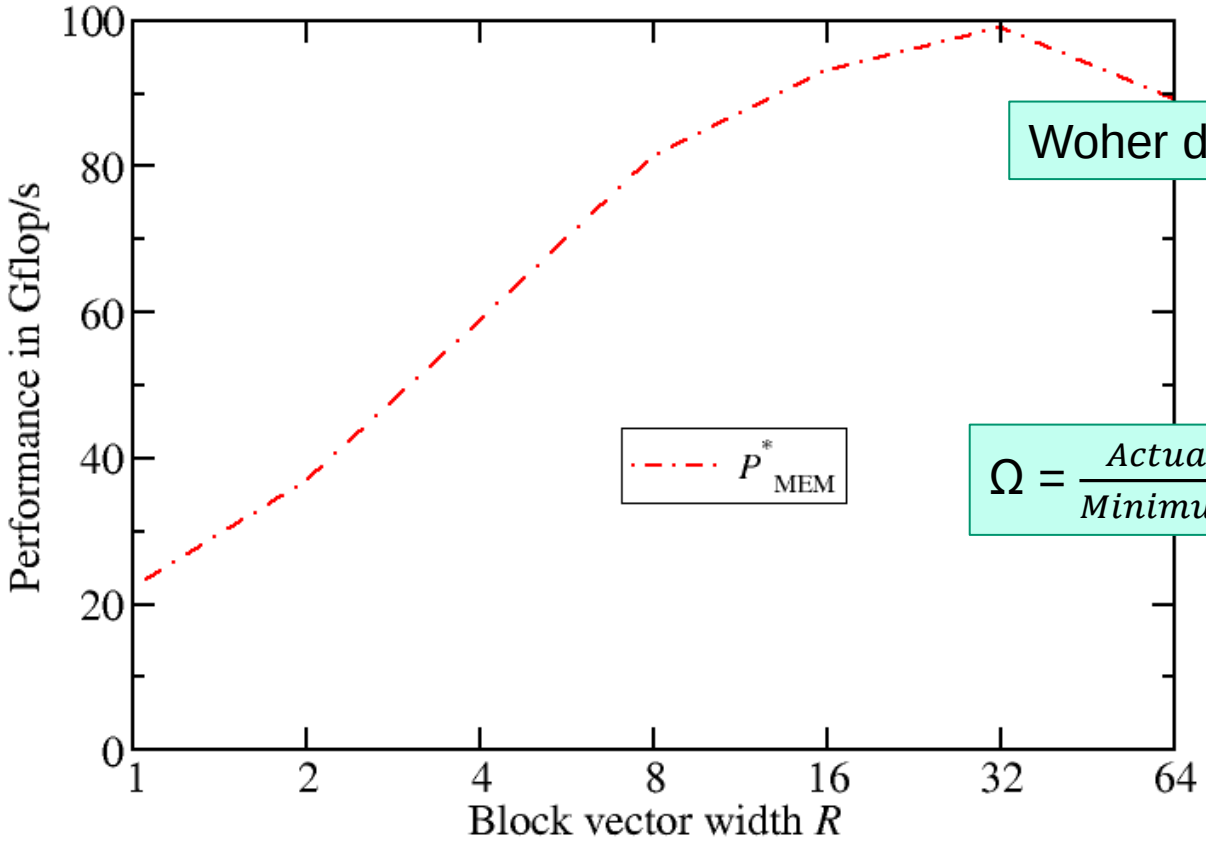
SNB: Intel Xeon Sandy Bridge, K20X: Nvidia Tesla K20X
Komplexe (Double Precision) Matrix/Vektoren (topologischer Isolator)

Performance Engineering

- Testen der Implementation in erster Linie gegen die Theorie
- Für Speichergebundene Operationen: Roofline Modell
- SpMVM: unregelmäßiger Speicherzugriff
  - Parametrisiertes Modell
  - Performance Counter nutzen (z.B. LIKWID Tool) um tatsächliches Datenvolumen zu messen



Roofline Analyse für spMMVM



Woher die Abnahme?

$$\Omega = \frac{Actual\ data\ transfers}{Minimum\ data\ transfers}$$

Übersicht

- Aktuelle Herausforderungen für High-End HPC
- Eigenwertprobleme: Anwendungen und Algorithmen
- Technologie: MPI+X, Fehlertoleranz, Nodelevel Performance
- **Software Entwicklungen: GHOST und PHIST**
- Fallbeispiele und Zusammenfassung

Software aus dem ESSEX Projekt

General, Hybrid, and Optimized Sparse Toolkit

- MPI + OpenMP + SIMD + CUDA
- Dünnbesetzte Matrix-(Block-)Vektor-Multiplikation
- Dichtbesetzte Block-Vektoroperationen
- (Einfache) Task-Queue für funktionale Parallelität
- Asychrones Checkpoint-Restart

Status: Experimentell (für „HPC affine“ C Programmierer)

http://bitbucket.org/essex/ghost

BSD Lizenz

Software aus dem ESSEX Projekt

PHIST **Pipelined Hybrid-parallel Iterative Solver** Toolkit

- Iterative Löser für dünnbesetzte Matrizen
  - Eigenwertprobleme: Jacobi-Davidson, FEAST
  - Lineare Gleichungssysteme (LGS): GMRES, MINRES, CARP-CG
- Einfaches funktionales Interface (C, Fortran, Python)
- Pipelining: Algorithmen für optimale Performance formuliert
- verschiedene Möglichkeiten der Einbindung in Anwendungen

Status: umfangreiches Testframework, teils experimentell

http://bitbucket.org/essex/phist

BSD Lizenz

DLR

Einbindung von PHIST in Ihre Anwendung

Wahl der “Kernel Bibliothek”

Benötigte Flexibilität

GHOST

Kein einfacher Zugriff auf Matrix Elemente

PHIST „builtin“

Nur CPU
F‘03+OpenMP
CRS Format

Trilinos

Versch. Arch.
Große C++ Code Base

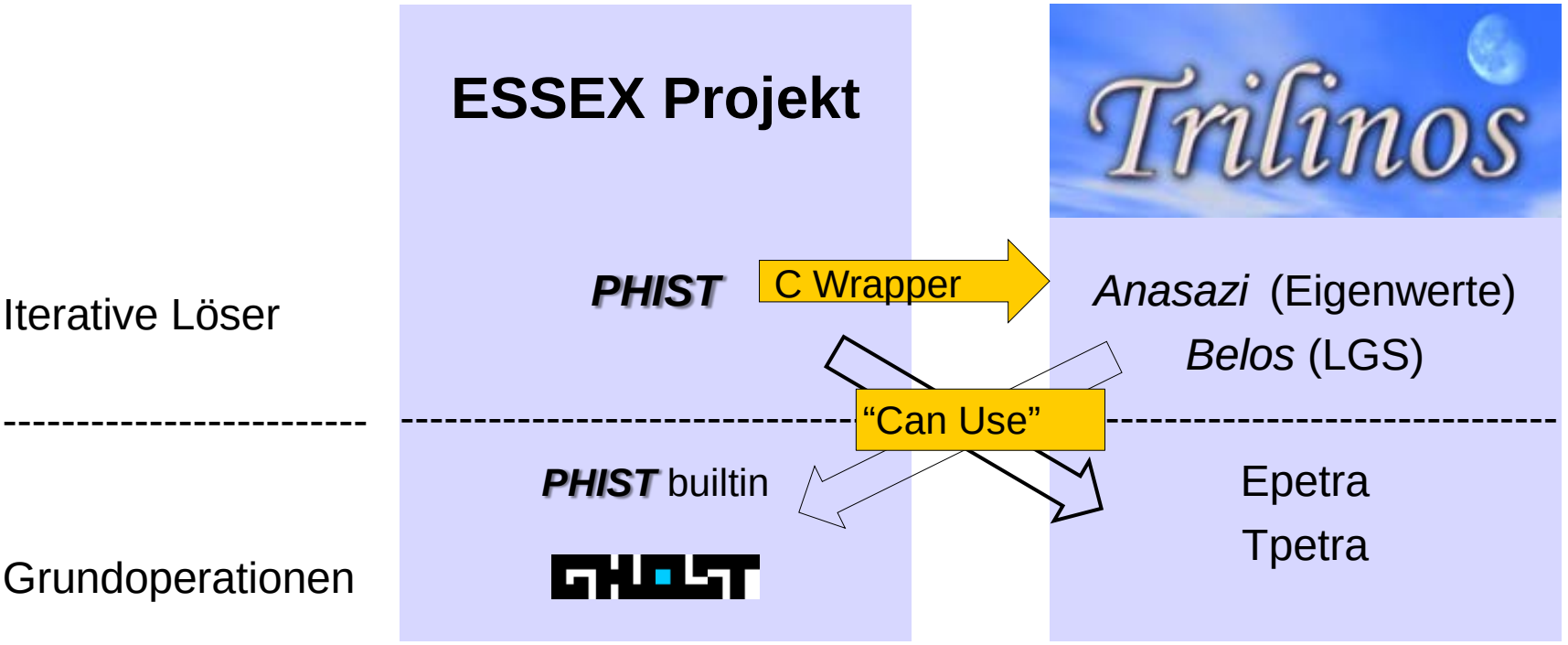
Eigene Datenstrukturen

Adapter ca 1000 Zeilen Code

Hardware-Nähe

DLR

Zweiseitige Interoperabilität von PHIST und Trilinos



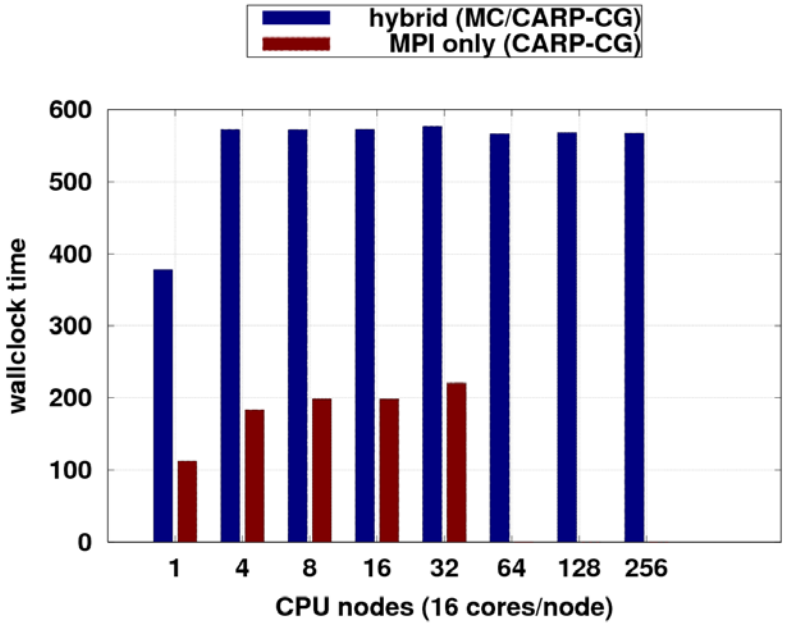
Übersicht

- Aktuelle Herausforderungen für High-End HPC
- Eigenwertprobleme: Anwendungen und Algorithmen
- Technologie: MPI+X, Fehlertoleranz, Nodelevel Performance
- Software Entwicklungen: GHOST und PHIST
- **Fallbeispiele und Zusammenfassung**

CARP-CG: Schwache Skalierung bis 5.1Mrd Unbekannte

- Zwei Varianten einer „row projection“ Methode
- Variante OpenMP nicht voll optimiert, aber offensichtlich speicher-effizienter

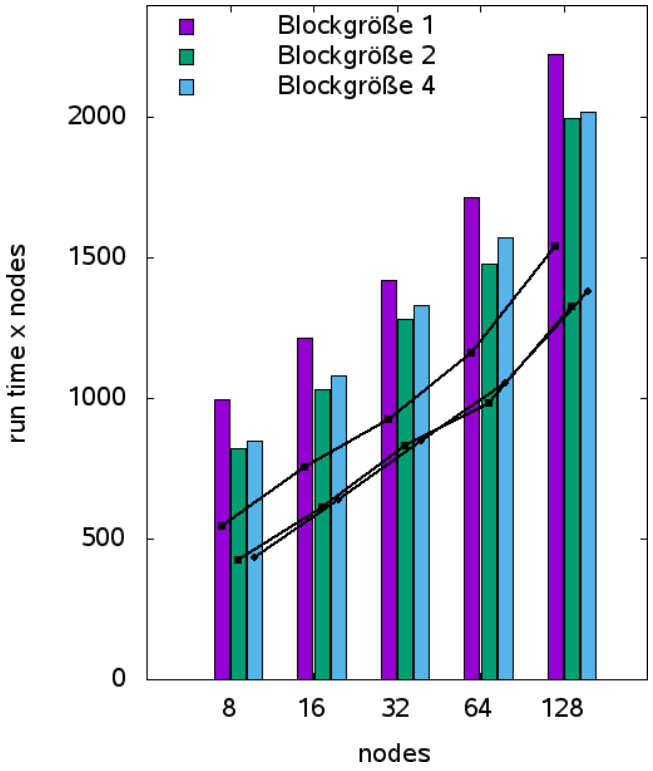
- **Anwendung:** „schwierige“ lineare Gleichungsysteme, z.B.
  - Helmholtz Gleichungen
  - Konvektionsdominierte Strömungen
  - Innere Eigenwertprobleme
  - Least Squares Probleme



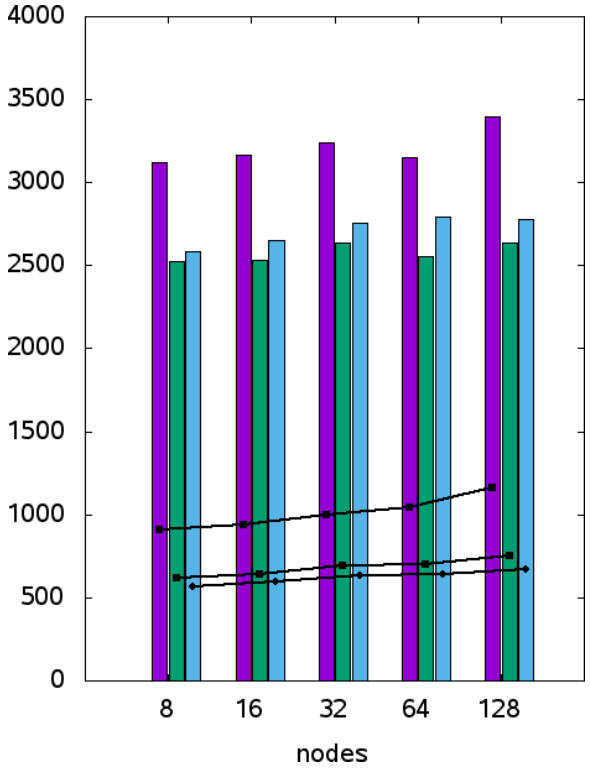
Beispiel: Graphene

Block Jacobi-Davidson, Starke Skalierung

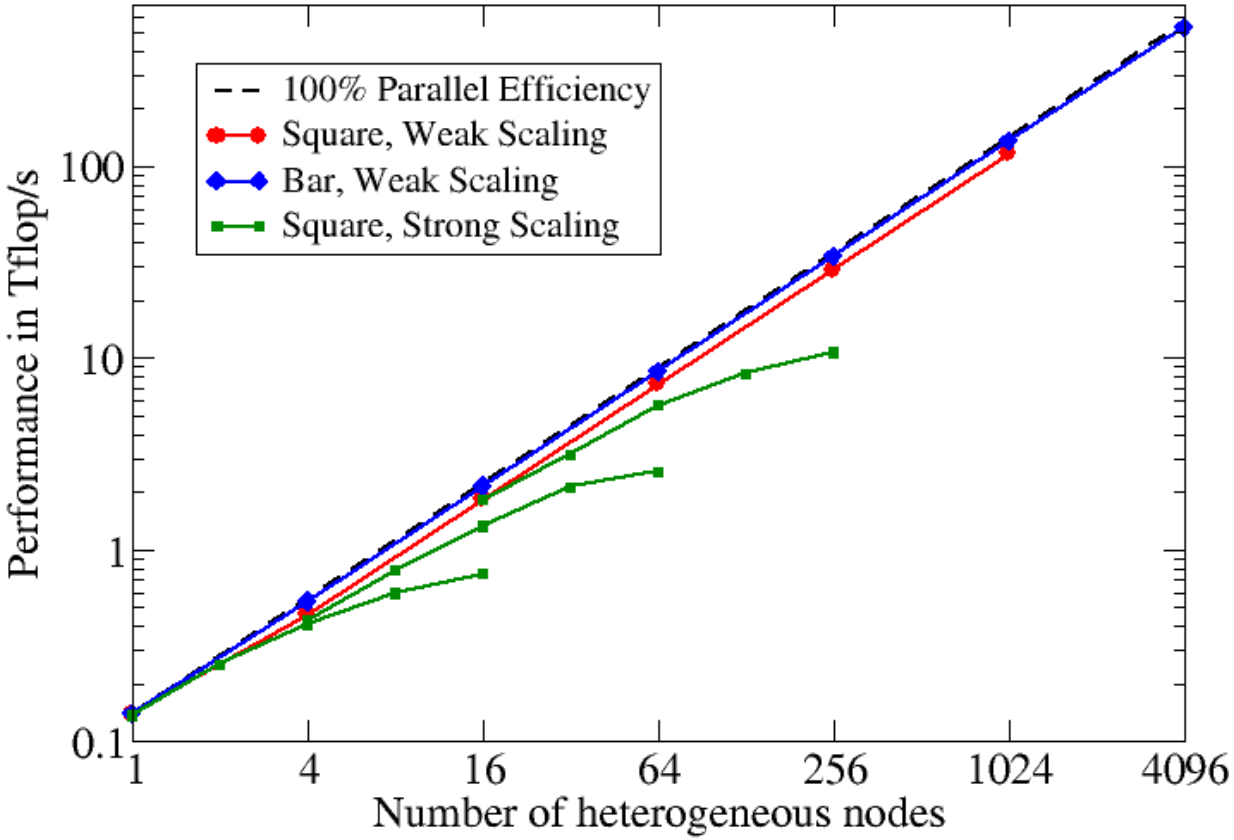
Spinkette



3D Konvektion-Diffusion



KPM, Skalierung auf heterogenem System



CRAY XC30 (*Piz Daint**)

- 5272 Rechenknoten mit jeweils
  - 1x 8-Kern Intel Sandy Bridge
  - 1x Nvidia Kepler K20x
- Peak: 7.8 Pflop/s
- System mit höchster Performanz in Europa

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

Zusammenfassung

- Skalierbarkeit fängt nicht erst bei einer CPU an
- GHOST und **PHIST** implementieren ein pragmatisches, Hardware-nahes Programmiermodell für heterogene Systeme
- Software mittels Performance Modellen verifiziert
- erhöhte Rechenintensität durch Blocking und Kernel Fusion
- ermöglicht ausfallsichere Anwendungen
- OpenSource: http://bitbucket.org/essex

DLR

Vielen Dank für ihre Aufmerksamkeit!

Fragen?

Dr. Jonas Thies

Simulations- und Softwaretechnik

Abt. Verteilte Systeme und Komponentensoftware

Jonas.Thies@dlr.de

http://www.DLR.de/sc

Dank an

Melven Röhrig-Zöllner (DLR)
Moritz Kreutzer und Faisal Shahzad (RRZE)

SPPEXA

DLR