A Software Infrastructure for Solving Quantum Physics Problems on Extremely Parallel Systems

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

- Research Institution
- Space Agency
- Project Management Agency
Approx. 8000 employees across 33 institutes and facilities at 16 sites.

DLR Institute Simulation and Software Technology
Scientific Themes and Working Groups

- Intelligent and Distributed Systems
- Distributed Software Systems
- Software Engineering
- Embedded Systems
- Modeling and Simulation
- Scientific Visualization
- 3D Interaction
- High-Performance Computing
- Software for Space Systems and Interactive Visualization
- Departments
- Working Groups
Survey

• ESSEX motivation

• The ESSEX software infrastructure

• Holistic view: application, algorithm and performance

• Conclusions

• Future work
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 applications:
- Graphene,
- topological insulators,
ESSEX: Physical Motivation and Sparse Eigenvalue problem

Solve large sparse eigenvalue problem

$$H x = \lambda x$$

$$i\hbar \frac{\partial}{\partial t} \psi(\vec{r}, t) = H \psi(\vec{r}, t)$$
ESSEX Software Development: Basics

- **Git** for distributed software development

- **Merge-request workflow** for code review; changes only in branches

- Own MPI extension for **Google Test**

- Realization of **continuous-integration** with Jenkins server
The ESSEX Software Infrastructure: Kernel Library (General Hybrid and Optimized Sparse Toolkit) provides

- intelligent resource management for heterogeneous 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

- System with multiple CPUs (NUMA domains) and GPUs
The ESSEX Software Infrastructure: MPI + X with

- System with multiple CPUs (NUMA domains) and GPUs
- -np 1: use entire CPU
The ESSEX Software Infrastructure: MPI + X with

- 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

- 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

- 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 \texttt{GHYST}
- holistic performance engineering
- portability and interoperability
The ESSEX Software Infrastructure: Test-Driven Algorithm Development

new algorithm

Algorithms
implement template
missing kernels
add unit tests
optimize numerics

Comp. Core
add robust kernels
implement optimized version
evaluate overall performance

established kernel library
optimized kernel library

application
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)

Required flexibility

PHIST „builtin“

CPU only
F’03+OpenMP
CRS format

Trilinos

Various arch.
large C++
code base

own
datastructures

Adapter ca 1000
lines of code

Hardware awareness

GHOST

No easy access
to matrix
elements
Interoperability of PHIST and Trilinos

**ESSEX project**

- **PHIST**
  - C Wrapper
  - "Can Use"

- **Anasazi** (eigenproblems)
  - **Belos** (lin. eq. syst.)

- **EPetra**
- **Tpetra**

**Iterative solvers**

**Basic operations**
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
    \[ Y = \alpha AX + \beta Y \text{ and } Y^T X \]
  • 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 $X = I$)

$$X(\omega) = \frac{1}{N} \text{tr}[\delta(\omega - H)X] = \frac{1}{N} \sum_{n=1}^{N} \delta(\omega - E_n) \langle \psi_n, X \psi_n \rangle$$
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⟩
    η_{2m} ← ⟨v|v⟩
    η_{2m+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

\[ |v\rangle \leftarrow |\text{rand()}\rangle \]

Initialization steps and computation of $\eta_0, \eta_1$

for $m = 1$ to $M/2$ do

\[ \text{swap}(\langle w\rangle, |v\rangle) \]

\[ |u\rangle \leftarrow H|v\rangle \]

\[ |u\rangle \leftarrow |u\rangle - b|v\rangle \]

\[ |w\rangle \leftarrow -|w\rangle \]

\[ |w\rangle \leftarrow |w\rangle + 2a|u\rangle \]

\[ \eta_{2m} \leftarrow \langle v|v\rangle \]

\[ \eta_{2m+1} \leftarrow \langle w|v\rangle \]

end for

for $r = 0$ to $R - 1$ do

\[ |v\rangle \leftarrow |\text{rand()}\rangle \]

Initialization steps and computation of $\eta_0, \eta_1$

for $m = 1$ to $M/2$ do

\[ \text{swap}(\langle w\rangle, |v\rangle) \]

\[ |w\rangle = 2a(H - b1)|v\rangle - |w\rangle \]

\[ \eta_{2m} = \langle v|v\rangle \]

\[ \eta_{2m+1} = \langle w|v\rangle \]

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:

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

▷ Augmented Sparse Matrix

```plaintext
|V⟩ := |v⟩₀...R - 1
|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⟩ = 2α(H - b1)|V⟩ - |W⟩ &
    η₂m[: ] = ⟨V|V⟩ &
    η₂m+1[: ] = ⟨W|V⟩ &
end for
```

▷ Multiple Vector Multiply

Augmented Sparse Matrix
KPM: Heterogenous Node Performance

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

Performance in Gflop/s

- SNB
- K20X
- SNB+K20X

Heterogeneous efficiency

Intel Xeon E5-2670 (SNB)
NVIDIA K20X

85% 90% 87%
KPM: Large Scale Heterogenous Node Performance

Performance Engineering of the Kernel Polynomial Method on Large-Scale CPU-GPU Systems

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

• Holistic performance engineering strategies successful for developing highly scalable solutions, cf. KPM.

• **PHIST** with **GHOLST** 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

• First convincing results with quantum physics applications
Future Work: Programming

Building Blocks, Parallelization, and Performance Engineering
- Holistic performance and power engineering
- Advanced building blocks engineering

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

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

Thanks to all partners from the ESSEX project and to DFG for the support through the Priority Programme 1648 “Software for Exascale Computing”.

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

International contacts

- Sandia (Trilinos project)
- Tennessee (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 High-Performance Computing
Achim.Basermann@dlr.de
http://www.DLR.de/sc