Software Sustainability in the Many-Core Era

Jonas Thies
German Aerospace Center (DLR)

Aerospace center, project manager and space agency

- > 8 000 employees
- 16(?) sites in Germany

Main areas of research

- Aeronautics
- Energy
- Space
- Security

For the ESA mission ‘Rosetta’, DLR developed and operates the ‘Philae’ lander

... so who am I to talk to you about software and HPC?
Institute Simulation and Software Technology

Software is developed everywhere at DLR

- 2005: ~ 25% of personnel expenses spent on software development
- cost: > 100 million Euro/year
- Examples: CFD, material science, onboard computers, data analysis...

Our mission (~ 50 staff) is to increase the efficiency of software development in other institutes by **software research**, **teaching** and contributing to **key projects**.
Equipping Sparse Solvers for the EXa-scale
Sparse Eigenvalue Problems

**Formulation** Find some Eigenpairs \((\lambda_j, v_j)\) of a large and sparse matrix (pair) in a target region of the spectrum

\[ \mathbf{A} v_j = \lambda_j \mathbf{B} v_j \]

- **A** Hermitian or general, real or complex
- **B** may be identity matrix (or not)
- ‘some’ may mean ‘quite a few’, 100-1 000 or so

**Applications**

- **Quantum and Fluid Mechanics**
  - Graphene
  - Hubbard model
  - Driven cavity
  - Rayleigh-Benard convection

**DLR applications**
Block Jacobi-Davidson QR

- Aim: partial QR decomposition, $A\hat{Q} = QR$, $R \in \mathbb{C}^{k \times k}$ upper triangular,

- $\frac{1}{2}Q^TQ - \frac{1}{2}I = 0$, $Q \in \mathbb{R}^{N \times k}$.

Newton’s method, let $Q = \tilde{Q} + \Delta Q$

- $A\Delta Q - \Delta Q\tilde{R} = A\tilde{Q} - \tilde{Q}\tilde{R}$

- $\tilde{Q}^T\Delta Q = 0$
Block Jacobi-Davidson QR (2)

This leads to a set of *correction equations*

$$(I - \bar{Q}\bar{Q}^T)A(I - \bar{Q}\bar{Q}^T)\Delta Q - \Delta \bar{Q}\bar{R} = A\bar{Q} - \bar{Q}\bar{R}$$

- Subspace acceleration: add corrections to expanding search space $V$
- Ritz-Galerkin: $M = V^T A V$, $M = S^H R S$
- Lock converged eigenpairs $\Rightarrow$ growing projection space $\bar{Q}$
- Solve correction eq. using (deflated) GMRES or MINRES Krylov solver
Projection-Based Eigensolvers

**Input:** Interval $I_\lambda$, Matrix pair $A, B \in \mathbb{C}^{N \times N}$

**Output:** $\hat{m}$ eigenpairs $(X, \Lambda)$ in $I_\lambda$

1. Estimate $\tilde{m} \approx \hat{m}$, choose random $Y \in \mathbb{C}^{N \times m}$ of rank $m > \tilde{m}$
2. **while** not $\tilde{m}$ pairs converged **do**
3. Compute $U = PY$ with suitable projector $P = P_{I_\lambda}(A, B)$
4. Compute Rayleigh quotients $A_U = U^*AU$ and $B_U = U^*BU$
5. Update estimate $\tilde{m}$ of $\hat{m}$ and adjust $m > \tilde{m}$
6. Solve EVP $A_UW = B_UW\Lambda$
7. $X \leftarrow UW$
8. Orthogonalize $X$ against locked vectors, lock newly converged ones
9. $Y \leftarrow BX$
10. **end while**
Two Ways of Computing the Projector $U = PY$

**BEAST-C/FEAST:** contour integration of resolvent function

$$U := \frac{1}{2\pi i} \int_C (zB - A)^{-1} Bdz Y$$

- Requires solving many independent but hard linear systems

**Polynomial expansion**

- Chebyshev iteration
- Requires very large number of spMMVMs
- But no global synchronization
- ‘Filter polynomials’ to reduce Gibbs oscillations

*aaka* ChebFD
Common Operations of Iterative Methods

1. Memory-bounded linear operations involving

- sparse matrices $A \in \mathbb{R}^{N \times N}$ (sparseMat)
- multi-vectors $X, Y \in \mathbb{R}^{N \times m}$ (mVecs)

Developed in ESSEX/\textbf{GHOST} (e.g. $Y \leftarrow \alpha AX + \beta Y$, $C \leftarrow X^T Y$, $X \leftarrow Y \cdot C$)

2. Algorithms for sdMats

- e.g. eigendecomposition of projected matrix
- use LAPACK/PLASMA/MAGMA

3. Sparse matrix (I)LU factorization

- not available in \textbf{GHOST}
- allow using external libraries via Trilinos interface
Comparing Performance Results

**simple(?) operation:** \( C = V^T V, V \in \mathbb{R}^{1M \times 4} \)
Comparing Performance Results

**simple(?) operation:** \( C = V^T V, V \in \mathbb{R}^{1M \times 4} \)
Comparing Performance Results

**simple(?) operation:** \( C = V^T V, \ V \in \mathbb{R}^{1M \times 4} \)
Comparing Performance Results

simple(?) operation: \( C = V^T V, \; V \in \mathbb{R}^{1M \times 4} \)

[Diagram showing performance results across different core counts]
Present Challenges to HPC Users

Performance increase on low to intermediate levels

- SIMD/SIMT
- Increasing core count
- Increasingly non-uniform cache/memory hierarchies

Many programming models and (semi-)standards

- OpenMP+OpenACC vs. OpenCL
- Vendor-specific (e.g. CUDA)
- Ca. 15 different tasking runtimes
- C++11, Intel TBB, Kokkos
- MPI vs. PGAS (GPI/GASPI, Co-Array Fortran, UPC)

imo: MPI is here to stay, the node-level is uncertain
Our Test System

- 2 × 12 core Haswell EP @2.3 GHz
- Theoretical Peak: 442 GFlop/s
- 128 GB RAM
- STREAM-Triad: 42 GB/s / socket

- Tesla K40 GPU
- Theoretical Peak: 1.43 TFlop/s
- 12 GB RAM
- STREAM-Triad: 215 GB/s
SPMD/OK Programming Model

- SPMD (‘BSP’) vs. task parallelism
- Heterogenous cluster: distribute problem according to limiting resource (e.g. memory bandwidth)
- Optimized Kernels make sure each component runs as fast as possible
- User sees a simple functional interface (no general-purpose looping constructs etc.)

A success story: Chebyshev methods on Piz Daint

Only needs sparse matrix times multiple vector (spMMV) products and an occasional vector operation
Upcoming Challenges

(even more) heterogeneous memory

- Knight’s Landing: additional fast NUMA domain
- IBM Power 9 + Nvidia Volta: GPU can read from main memory (at same speed as CPU)

Algorithm developer must decide which data should be accessed fast

- E.g. eigensolvers often have an outer/inner (project/correct) structure, the complete outer search space may not be needed in inner loop
**PHIST Software Architecture**

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

- facilitate algorithm development using **GHST**
- holistic performance engineering
- portability and interoperability
**PHIST** Software Architecture

a Pipelined Hybrid-parallel Iterative Solver Toolkit

- facilitate algorithm development using **GHOST**
- holistic performance engineering
- portability and interoperability
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)
Cool Features of *PHIST* and *GHOST*

**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
The Test-Driven HPC Development Process

Nightly **PHIST** runs with thousands of unit tests for various

- #MPI procs, #threads
- data types (S/D/C/Z)
- block sizes and memory alignment
- vectorization (SSE,AVX,CUDA)

![Diagram of the test-driven HPC development process]

- **Algorithms**
  - Implement template
  - Missing kernels
  - Add unit tests
  - Optimize numerics
  - New algorithm

- **Comp. Core**
  - Add robust kernels
  - Implement optimized version
  - Evaluate overall performance

- Established kernel library
- Optimized kernel library

Application
Block Vector Operations on CPU and GPU

Block vector inner product
\[ C = V^T W, \quad W \in \mathbb{R}^{N \times 4} \]

Reductions don’t hurt that much!

\[ V_{1:m/2} \cdot C \quad (used \ to \ shrink \ basis \ in \ iterative \ methods) \]

Should avoid larger block sizes...
Proposed Data Layout for Large Blocks

Replace mVec data type by array of ghost_densemats

- need to update **GHOST** adaptor in **PHIST**
- no adjustments in kernels, interface, or algorithms needed
- unit tests will ensure correctness of refactoring
- perfcheck will reveal performance benefits
Proposed Data Layout for Large Blocks

Replace mVec data type by array of ghost_dense matrices

- need to update **GHOST** adaptor in **PHIST**
- no adjustments in kernels, interface, or algorithms needed
- unit tests will ensure correctness of refactoring
- perfcheck will reveal performance benefits

estimated effort:

**PHIST /BEAST** 2-3 weeks  
**FEAST, z-Pares** impossible  
**Anasazi** 2-3 months  
**SLEPc** impossible  
**PARPACK, PRIMME** 2-3 years
Summary: Sustainable HPC Software

**Good programming practice**

- Kernels and data structures belong together (object-oriented programming)
- Separation of kernels, core and high-level algorithms through interfaces
- Which are verified by tests, benchmarks and performance models

**Bad programming practice**

- Interfaces that expose raw data (e.g. reverse communication)
Questions?

Contact

Jonas Thies

DLR Simulation and Software Technology
High Performance Computing

Jonas.Thies@DLR.de
Phone 02203 / 601 41 45
http://www.DLR.de/sc

Links

► Project website
http://blogs.fau.de/essex/

► Source code
https://bitbucket.org/essex/