Software and Performance Engineering for Iterative Eigensolvers

Jonas Thies

German Aerospace Center (DLR)
Simulation and Software Technology
High Performance Computing

project ESSEX
Motivation 1: analyze nonlinear PDE systems

$2^{nd}$ order PDE after space discretization

\[ M \frac{\partial \Phi}{\partial t} = F(\Phi, t) \]

- with suitable boundary and initial conditions

Steady state; $\Phi$ as $t \to \infty$.

Standard technique: time stepping

- may take very long
- no information about stability

Physical difficulty: low frequency modes affect solution on very long time scales

Example: 3D Boussinesq equations

\[
\begin{align*}
\frac{\partial u}{\partial t} &= -((uu)_x + (vw)_y + (wu)_z) - p_x + \nu \nabla^2 u \\
\frac{\partial v}{\partial t} &= -((uv)_x + (vv)_y + (wv)_z) - p_y + \nu \nabla^2 v \\
\frac{\partial w}{\partial t} &= -((uw)_x + (vw)_y + (ww)_z) - p_z + \nu \nabla^2 w + g \alpha T \\
\frac{\partial T}{\partial t} &= -((uT)_x + (vT)_y + (wT)_z) + \kappa \nabla^2 T \\

u_x + v_y + w_z &= 0
\end{align*}
\]
Motivation 1: analyze nonlinear PDE systems

\[ M \frac{\partial \Phi}{\partial t} = F(\Phi, t) \]

- with suitable boundary and initial conditions

Steady state; \( \Phi \) as \( t \to \infty \).

Standard technique: time stepping
  - may take very long
  - no information about stability

Physical difficulty: low frequency modes affect solution on very long time scales

Example: 3D Boussinesq equations

\[
\begin{align*}
\frac{\partial u}{\partial t} &= -((uu)_x + (vw)_y + (wu)_z) - p_x + \nu \nabla^2 u \\
\frac{\partial v}{\partial t} &= -((uv)_x + (vv)_y + (wv)_z) - p_y + \nu \nabla^2 v \\
\frac{\partial w}{\partial t} &= -((uw)_x + (vw)_y + (ww)_z) - p_z + \nu \nabla^2 w + g \alpha T \\
\frac{\partial T}{\partial t} &= -((uT)_x + (vT)_y + (wT)_z) + \kappa \nabla^2 T \\
\end{align*}
\]

\[ u_x + v_y + w_z = 0 \]

Our approach:

- Newton-Krylov with preconditioning
- ‘parameter continuation’ as globalization
- linear stability analysis \( \Rightarrow \) solve
  \[ A x = \lambda B x \] for some \( \lambda \)s near 0, 
  \( B \) spd, \( A \) not.
Example: Rayleigh-Bénard convection

- Cube-shaped domain
- heated from below
- Rayleigh-Number
  \[ Ra = \frac{\alpha g \Delta T d^3}{\nu \kappa} \]

Figure: Flow patterns near the first three primary bifurcations
(a) x/y roll,
(b) diagonal roll,
(c) four rolls,
(d) toroidal roll
Motivation 2: provide a useful solver library

(i) **Application scientists** miss solvers that ...
   - can handle generalized and non-Hermitian problems
   - can be integrated deeply into applications
   - can easily be used from Fortran
   - support GPU accelerators and heterogenous hardware

(ii) **Numericists** need a platform for
   - implementing algorithms on increasingly complex hardware
   - performing meaningful performance studies

(iii) **Portability requirements:**
   - easy testing and benchmarking on all levels
Jacobi-Davidson: Newton’s as an Eigensolver

- Eigenvalue problem: solve $Ax - \lambda x = 0$ for $(x, \lambda)$
- Apply inexact Newton
- JDQR: subspace acceleration, locking and restart (Fokkema’99)

Jacobi-Davidson correction equation

- current approximation: $A\tilde{v} - \tilde{\lambda}\tilde{v} = r$,
- previously converged Schur vectors $(q_1, \ldots, q_k) = Q$
- solve approximately $(A - \tilde{\lambda}I)\Delta v = -r$, $\Delta v \perp \tilde{Q} = (Q, \tilde{v})$
- use some steps of preconditioned GMRES

Implementation: https://bitbucket.org/essex/phist
Block JDQR

**outer loop:** work on $n_b$ Ritz values $\tilde{\lambda}_j$ at a time
**Inner solver:** compute $t_j \perp \tilde{Q}$

without preconditioning:

\[ P(A - \tilde{\lambda}_j I)t_j = -r_j \]
\[ P = (I - \tilde{Q} \tilde{Q}^T) \]

with (left) preconditioning,

\[ P_K K^{-1}(A - \tilde{\lambda}_j I)t_j = -P_K K^{-1}r_j \]
\[ P_K = (I - \tilde{Q}_K (\tilde{Q}^T \tilde{Q}_K)^{-1} \tilde{Q}^T) \]

where $K$ is a preconditioner for $A - \tilde{\lambda} I$ and $\tilde{Q}_K = K^{-1} \tilde{Q}$.

**blocked solvers:** separate Krylov spaces, but using block kernels.
**outer loop:** orthogonalize $t_j$ against $[Q, V]$, expand $V$. 

---

[DLR Logo]
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/\texttt{GH\textsc{ost}} (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
- \texttt{LAPACK}/\texttt{PLASMA}/\texttt{MAGMA}

3. Sparse matrix (I)LU factorization

- not available in \texttt{GH\textsc{ost}}
- allow using external libraries via \texttt{Trilinos} interface
Why do we need our own kernels?

simple(?) operation: \( C = V^T V, V \in \mathbb{R}^{1M \times 4} \)
Why do we need our own kernels?

**simple(?) operation:** $C = V^T V, \ V \in \mathbb{R}^{1M \times 4}$
Why do we need our own kernels?

**simple(?) operation:** \( C = V^T V, \ V \in \mathbb{R}^{1M \times 4} \)
Why do we need our own kernels?

simple(?) operation: $C = V^T V$, $V \in \mathbb{R}^{1M \times 4}$
**SPMD/OK Programming Model**

- SPMD (‘BSP’) vs. task parallelism
- Heterogeneous 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.
PHIST software architecture

a Pipelined Hybrid-parallel Iterative Solver Toolkit

- facilitate algorithm development using **GHIST**
- 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)
PHIST interface example

**Inspired by MPI:** objects represented by handles only

**C/C++:**

```c
// compute y = alpha*A*x + beta*y
void phist_DsparseMat_times_mvec(double alpha, phist_Dconst_sparseMat_ptr A,
                                  phist_Dconst_mvec_ptr x, double beta, phist_Dmvec_ptr y, int* iflag);
```

**Fortran 2003:**

```fortran
subroutine phist_DsparseMat_times_mvec(alpha, A, x, beta, y, iflag)
  use iso_c_binding, only: c_double, c_ptr, c_int
  use phist_types
  real(c_double), value :: alpha, beta
  type(Dconst_sparseMat_ptr), value :: A
  type(Dconst_mvec_ptr), value :: x
  type(Dmvec_ptr), value :: y
  integer(c_int) :: iflag
```

similar **Python** interface exists

**Inspired by Petra:** comm, map, views
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
Example application: Turing problem

Reaction-Diffusion problem

\[
\frac{\partial u}{\partial t} = D \delta \nabla^2 u + \alpha u(1 - r_1 v^2) + v(1 - r_2 u) \\
\frac{\partial v}{\partial t} = \delta \nabla^2 v + v(\beta + \alpha r_1 uv) + u(\gamma + r_2 v) 
\]

(1)

- 2D: spot and stripe patterns
- can be solved using AMG
- non-normality: JDQR + AMG fails!
3D Turing: many patterns and bifurcations
3D Turing: many patterns and bifurcations
Preconditioning may be dangerous...

(normalized) projected operator \( V^T P_K K^{-1} A V \) after 150 Arnoldi iterations

with 1 eigenvector of \( A \) in \( P_K \)

We used an adaptation of Trefethens Matlab code: http://www.cs.ox.ac.uk/pseudospectra/software.html
Preconditioning may be dangerous...

(normalized) projected operator $V^T P_K K^{-1} A V$ after 150 Arnoldi iterations

with 5x eigenvectors of $A$ in $P_K$

We used an adaptation of Trefethen's Matlab code:
http://www.cs.ox.ac.uk/pseudospectra/software.html
Turing with preconditioning

To avoid introducing non-normality by an ill-conditioned preconditioner, use AMG (ML) on the Laplacian:

**Number of BJDQR(4) iterations**

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>GMRES</th>
<th>GMRES + AMG</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**solve time**
(weak scaling on 8, 64 and 512 cores)

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Time [s]</th>
<th>ML (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>65%</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>67%</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>79%</td>
<td></td>
</tr>
</tbody>
</table>
Performance portability with PHIST+GHOST

- Find 20 left-most eigenpairs of a spin-chain matrix \((N \approx 2.7M)\)
- BJDQR + MINRES
- Run time determined by main memory bandwidth

![Chart](attachment:chart.png)
Scaling on Piz Daint

- 3D non-symmetric PDE problem
- block Jacobi-Davidson + GMRES
- find 10 right-most eigenvalues

It’s like hungry beasts feeding from very small plates
Summary: do we provide a useful solver library?

(i) PHIST...
- can handle generalized and non-Hermitian problems (with caveats)
- can be integrated deeply into applications by exposing the kernel interface
- can easily be used from Fortran via Fortran bindings in phist_fort and builtin Fortran kernels
- supports GPU accelerators and heterogeneous hardware via GHOST and allows Numericists to
  - implement algorithms using an abstract interface to GHOST and other libraries
  - compare algorithms using the same backend
  - and backends with the same algorithm
(ii) Portable and maintainable
- ~10 000 test cases for kernels, core and algorithms (make test)
- perfcheck: report roofline performance of kernels after solver run
Future Work

- more memory-efficient variant for GPUs
  - do not store $AV$
  - use QMR instead of GMRES)
- more interoperability
  - e.g. apply Trilinos preconditioner to GHOST vector
- better understanding of non-Hermitian problems annd preconditioning
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/

Joint work with the group of Gerhard Wellein (U. Erlangen) and Fred Wubs (U. Groningen).
Funding was provided by DFG priority programme 1648 (SPPEXA) project ESSEX.