Linear algebra tasks in Materials Science: optimization and portability

ADAC Workshop, July 17-19 2017 | Edoardo Di Napoli
Outline

Jülich Supercomputing Center

Chebyshev Accelerated Subspace Iteration (ChASE)

Hamiltonian and Overlap matrices initialization in FLAPW
Forschungszentrum Jülich
Jülich Supercomputing Centre

Supercomputing operations for:
- Center: FZJ
- Region: JARA
- Germany: Gauss Centre for Supercomputing
  John von Neumann Institute for Computing
- Europe: PRACE, EU projects

Application support
- Primary support & Simulation Labs
- Peer review support & coordination

R&D work
- Methods and algorithms, computational science, performance analysis and tools
- Scientific Big Data Analytics with HPC
- Computer architectures, Co-Design Exascale Labs together with IBM, Intel, NVIDIA

Education and training
Supercomputing systems: dual architecture strategy

IBM Power 4+ JUMP, 9 TFlop/s

IBM Power 6
JUMP, 9 TFlop/s

IBM Blue Gene/L
JUBL, 45 TFlop/s

Intel Nehalem
JUROPA
300 TFlop/s

IBM Blue Gene/P
JUGENE, 1 PFlop/s

File Server

Intel Haswell
JURECA
~ 2.2 PFlop/s

+ Booster
~ 10 PFlop/s

IBM Blue Gene/Q
JUQUEEN
5.9 PFlop/s

Lustre
GPFS

2014

JUQUEEN successor
~ 50 PFlop/s

2019

General-Purpose Cluster

Highly Scalable System
Simulation Laboratory as HPC enabler

- **Advisory Board**
- **Community Groups**
- **Simulation Laboratory**
  - **Support:**
    - Application analysis
    - Re-engineering
    - Community codes
    - Workshops
  - **Research:**
    - Scalable algorithms
    - XXL simulations
    - 3rd party projects
    - Hardware co-design
- **Cross-Sectional Teams, Exascale and Data Laboratories**
SimLab Quantum Materials (SLQM)

Mission

The Simulation Laboratory Quantum Materials (SLQM) provides expertise in the field of quantum-based simulations in Materials Science with a special focus on high-performance computing. SLQM acts as a high-level support structure in dedicated projects and hosts research projects dealing with fundamental aspects of code development, algorithmic optimization, and performance improvement. The Lab acts as an enabler of large scale simulations on current HPC platforms as well as on future architectures by targeting domain-specific co-design processes.
High-performance computations

Observations:

- Linear algebra algorithms ubiquitous in *Materials Science*
- Numerical libraries are used as *black boxes*.
- Domain-specific *knowledge* does not influence algorithm choice.
- Rigid legacy codes are hard to *modernize*.

**Goal**

*Design and optimize* linear algebra algorithms in order to:

- exploit available *knowledge*.
- increase the *parallelism* of complex tasks.
- facilitate *performance portability*
SLQM active areas of research

- **Adaptive integration** for functional Renormalization Group (fRG)
- HPC **Tensor** operations (contractions, generalized contractions, transpositions, etc.) with application in fRG, Quantum Chemistry, etc.
- Algebraic **eigensolvers** (sequences of dense eigenproblems, sparse eigenproblems) with applications in Density Functional Theory (DFT), Numerical RG, Excitonic Hamiltonians, etc.
- Articulated **linear algebra tasks** (Matrix initialization, etc.)
- Structured **Linear Systems** (Dyson, Poisson, and Keldish equations)
- Self-consistent Field **preconditioners**
- Predicting material properties through **machine learning**
Two examples from FLAPW methods (DFT)

An eigensolver for sequences of dense matrices

- **Chebyshev Accelerated Subspace iteration Eigensolver (ChASE)** – Exterior iterative eigensolver tailored to sequences of *dense* Hermitian eigenvalue problems arising in DFT methods based on the LAPW basis set.

A complex linear algebra task

- **$H$ and $S$ Dense Linear Algebra (HSDLA) algorithm** – Combination of dense linear algebra kernels for the initialization of Hamiltonian and Overlap matrices in Density Functional Theory.
Density Functional Theory Self-Consistent Field

General framework

- **Initial guess** for charge density $n_{\text{start}}(r)$

- **Initialize** $A_k^{(\ell)}$ and $B_k^{(\ell)}$ matrices

- **Solve a set of eigenproblems** $P_{k_1}^{(\ell)} \ldots P_{k_N}^{(\ell)}$

- **Converged?** $|n^{(\ell)} - n^{(\ell-1)}| < \eta$

- **No**

- **Yes**

- **Compute new charge density** $n^{(\ell)}(r)$

- **OUTPUT** Electronic structure, ...
The case of the FLAPW method

Observations:

1. every $P^{(\ell)}_k : A^{(\ell)}_k x = B^{(\ell)}_k \lambda x$ is a generalized eigenvalue problem;
2. $A$ and $B$ are hermitian ($B$ is positive definite);
3. required: lower $2 \div 20\%$ of eigenpairs;
4. eigenvectors of problems of same $k$ are seemingly uncorrelated across iterations $i$;
5. k-vector index: $k = 1 : 10 \div 100$;
6. iteration cycle index: $\ell = 1 : 20 \div 50$. 
Outline

Jülich Supercomputing Center

Chebyshev Accelerated Subspace Iteration (ChASE)

Hamiltonian and Overlap matrices initialization in FLAPW
Two distinct optimization opportunities:

1. **Knowledge exploitation**: solution of previous SCF cycle can be used to “precondition” the solver at the next iteration.

2. **Algorithm optimization**: polynomial degree can be pre-computed in order to minimize Mat-Vec multiplications.
Chebyshev Accelerated Subspace Eigensolver

Chebyshev filter

A generic vector \( v = \sum_{i=1}^{n} s_i x_i \) is very quickly aligned in the direction of the eigenvector corresponding to the extremal eigenvalue \( \lambda_1 \)

\[
v^m = p_m(A)v = \sum_{i=1}^{n} s_i p_m(A)x_i = \sum_{i=1}^{n} s_i p_m(\lambda_i)x_i = s_1 x_1 + \sum_{i=2}^{n} s_i \frac{C_m(\lambda_i - c)}{e} x_i \sim s_1 x_1
\]
The core of the algorithm: Chebyshev filter

> 80% Computing time

Three-terms recurrence relation

\[ C_{m+1}(t) = 2xC_m(t) - C_{m-1}(t); \quad m \in \mathbb{N}, \quad C_0(t) = 1, \quad C_1(t) = x \]

\[ Z_m = p_m(\tilde{H}) Z_0 \quad \text{with} \quad \tilde{H} = H - cI_n \]

**FOR:** \( i = 1 \rightarrow \text{DEG} - 1 

\[ Z_{i+1} \leftarrow 2 \frac{\sigma_{i+1}}{e} \]

\[ \tilde{H} \times Z_i - \sigma_{i+1} \sigma_i \]

\[ Z_{i-1} \times \text{GEMM} \]

**END FOR.**
Preconditioning ChASE

Exploiting knowledge

\[ X \equiv \{x_1, \ldots, x_n\} \]

\[ \Lambda \equiv \text{diag}(\lambda_1, \ldots, \lambda_n) \]
Speed-up

Speed-up = \frac{\text{CPU time (input random vectors)}}{\text{CPU time (input approximate eigenvectors)}}

\text{Au}_{98}\text{Ag}_{10} - n = 13,379 - 128 \text{ cores.}
ChASE with polynomial optimization

**Speed-up**

$$\text{Speed-up} = \frac{\text{CPU time (} m \text{ constant)}}{\text{CPU time (} m \text{ optimized for each vector)}}$$

![Graph showing speed-up comparison between 24 cores with multi-threaded BLAS and 24 cores with 1 GPU using CUBLAS.](image)
ChASE pseudocode

**INPUT:** Hamiltonian $H$, TOL, DEG — OPTIONAL: approximate eigenvectors $Z_0$, extreme eigenvalues $\{\lambda_1, \lambda_{\text{NEV}}\}$.

**OUTPUT:** NEV wanted eigenpairs $(\Lambda, W)$.

1. **Lanczos DoS step.** Identify the bounds for the *eigenspectrum interval* corresponding to the wanted eigenspace.

**REPEAT UNTIL CONVERGENCE:**

2. **Chebyshev filter.** Filter a block of vectors $W \leftarrow Z_0$.
3. Re-orthogonalize the vectors outputted by the filter; $W = QR$.
4. Compute the *Rayleigh quotient* $G = Q^\dagger HQ$.
5. Compute the primitive Ritz pairs $(\Lambda, Y)$ by solving for $GY = Y\Lambda$.
6. Compute the approximate Ritz pairs $(\Lambda, W \leftarrow QY)$.
7. **Check** which one among the Ritz vectors *converged*.
8. **Deflate** and *lock* the converged vectors.

**END REPEAT**
Execution time for distinct platforms
An example from Density Functional Theory
ChASE new implementation

**C++ Library**

- Separation between algorithm and implementation
  - Few kernels for low-level tasks (Lanczos, QR, Mat-Mat multiply, etc.)
  - General interface
  - Kernels adapted to the parallel computing platform
  - Library templated for Real, Complex, SP and DP.

**Integration into applications**

- FLEUR code
- Jena BSE code (based on VASP)
- early stage collaboration with Exciting developers
- early stage collaboration with developers from AICS (Kobe)
Outline

Jülich Supercomputing Center

Chebyshev Accelerated Subspace Iteration (ChASE)

Hamiltonian and Overlap matrices initialization in FLAPW
Hamiltonian and Overlap matrices

Matrix form

\[ H = \sum_{a=1}^{N_A} A^H_a T_a^{[AA]} A_a + A^H_a T_a^{[AB]} B_a + B^H_a T_a^{[BA]} A_a + B^H_a T_a^{[BB]} B_a \]

\[ S = \sum_{a=1}^{N_A} A^H_a A_a + \sum_{a=1}^{N_A} B^H_a \hat{U}^H_a \hat{U} B_a \]

\( A_a \) and \( B_a \in \mathbb{C}^{N_L \times N_G} \) while \( T_a \in \mathbb{C}^{N_L \times N_L} \) and Hermitian.

\( N_L = (l_{\text{max}} + 1)^2 \leq 121 \), \( N_G = 1,000 \div 50,000 \), and \( N_A = \) number of atoms.
Constructing $S_{AA}$
An example of memory layout re-structuring

$$S_{AA} = \sum_{a=1}^{N_A} A^H_a A_a.$$

1: $\text{for } a := 1 \rightarrow N_A \text{ do}$
2: $S_{AA} = A^H_a A_a$
3: $\text{end for}$

\[ \triangleright \ (\text{zherk: } 4N_LN_G^2 \text{ Flops}) \]

1: $S_{AA} = A^*_A A_*$

\[ \triangleright \ (\text{zherk: } 4N_A N_L N_G^2 \text{ Flops}) \]
Constructing $H_{AB+BA+BB}$

An example of algorithm re-structuring

\[
H_{AB+BA+BB} = \sum_{a=1}^{N_A} B_a^H (T_a^{[BA]} A_a) + (A_a^H T_a^{[AB]}) B_a + \\
\frac{1}{2} B_a^H (T_a^{[BB]} B_a) + \frac{1}{2} (B_a^H T_a^{[BB]}) B_a
\]

\[
= \sum_{a=1}^{N_A} B_a^H (T_a^{[BA]} A_a + \frac{1}{2} T_a^{[BB]} B_a) + \\
(A_a^H T_a^{[AB]} + \frac{1}{2} B_a^H T_a^{[BB]}) B_a
\]

1: \textbf{for} a := 1 \rightarrow N_A \textbf{ do} \\
2: \quad Z_a = T_a^{[BA]} A_a \quad \triangleright (\text{zgemm: } 8N_L^2 N_G \text{ Flops}) \\
3: \quad Z_a = Z_a + \frac{1}{2} T_a^{[BB]} B_a \quad \triangleright (\text{zhemm: } 8N_L^2 N_G \text{ Flops}) \\
4: \quad \text{Stack } Z_a \text{ to } Z_* \text{ and } B_a \text{ to } B_* \\
5: \textbf{end for} \\
6: H = Z_*^H B_* + B_*^H Z_* \quad \triangleright (\text{zher2k: } 8N_A N_L N_G^2 \text{ Flops})
FLEUR early optimizations
Minimization of FLOPS in FLEUR

\[ S_{AA} = \frac{(4\pi)^2}{\Omega} \sum_a \exp\left[i (K_G - K_{G'}) \cdot x_a \right] \sum_{l=0}^{l_{sph}} \frac{f_l,a,G f_l,a,G'}{W_{l,a}^2} \]

The sum over \( m \) can be removed by using the well known identity

\[ P_l (\hat{K}_G \cdot \hat{K}_{G'}) \frac{2l + 1}{4\pi} = \star \]

Removing the sum over \( m \) yields a reduction in \#ops by a factor of \((l_{sph} + 1) \sim 10\) but destroys matrix form \( \Longrightarrow \) “premature optimization”
Hamiltonian and Overlap matrices initialization

Main features

- Boosted **performance** through highly optimized computational kernels.
- Performance **portability** through modular implementation.
- Compute bound implementation at the cost of **increased complexity** (more FLOPs).

![Graph](image.png)

Figure 1: Speedup of our algorithm over FLEUR with $k_{max} = 4$ and increasing parallelism
Porting to heterogeneous architectures
Back-of-the-envelope analysis

- 5 lines of the algorithm constitute 97% of flops
- Correspond to BLAS-3 operations (gemm, herk, her2k)
- High arithmetic intensity and should fit GPUs well

- First step: offload these routine calls
- All 5 are BLAS kernels. Can we use some library?
  - cuBLAS
  - cuBLAS-XT
  - MAGMA
  - BLASX
Porting to heterogeneous architectures

Additional code?

- 3x wrappers \textit{(zgemm, zherk, zher2k)}
- Init and cleanup of cuda runtime and devices
  - Get devices, initialize devices, create handlers, ...
  - Destroy handlers, free devices, ...
- Allocate data in page-locked memory
  - Avoid “hidden” copies
  - Fast data transfer

Only around 100 lines of additional code
Experimental results
Test case 2: AuAg \((N_A = 108, N_L = 121, N_G = [3275 − 13379])\)

Sandy Bridge: CPU: E5-2650, 2 x 8core, 2.0GHz, 64GBs RAM, 2 Nvidia Tesla K20X
Peak performance: 256 GFs/s + 2 x 1.3 TFs/s
Conclusions

- Modernizing algorithm structure of legacy code is critical
- Tailoring algorithms to application-specific tasks
- Exploiting knowledge from domain-specific applications
- Layered design built on top of standardized libraries
- Increase in performance
- (Almost) free lunch ⇒ performance portability
References


Thank you

For more information

e.di.napoli@fz-juelich.de

http://www.fz-juelich.de/ias/jsc/slqm