Performance Evolution of the FLEUR Code

U.Alekseeva, G.Michalicek and D. Wortmann

Peter Grünberg Institut and Institute for Advanced Simulation, Forschungszentrum Jülich and JARA, 52425 Jülich, Germany

JÜLICH Forschungszentrum

MAX DRIVING THE EXASCALE TRANSITION

Abstract

The advances in performance optimisation of the all-electron DFT code FLEUR achieved during the MaX project are presented. The improved implementation features increased modularity of the code, reduced I/O, hybrid MPI/OpenMP parallelisation, additional interfaces to external libraries providing performance portability and provides a significantly improved users experience. Due to the performance boost of the MaX version of the FLEUR code simulations with unit cells of more than 1000 atoms are now feasible.

Unique Set of Features

At PGI-1/IAS-1 we develop and use the advanced DFT-code FLEUR: www.flapw.de

fleur

- Highly accurate FLAPW method
- All electron, full-potential DFT
- Bulk systems, surfaces, molecules
- Complex magnetism, including non-collinear magnetism, calculation of model parameters for localized spin models
- Relativistic effects, spin-orbit coupling
- Applied external electric fields
- Standard DFT exchange and correlation functionals as well as LDA/GGA+U, hybrid functionals, vdW functionals
- Interfaces to external codes like Wannier90, SPEX, GFLEUR

Algorithms for FLAPW

A standard FLEUR run performs the following main tasks:

1. The calculation of the potentials from the charge density
   - Coulomb potential as solution of Poisson equation
   - Exchange-correlation potential
2. The setup of the Hamiltonian and overlap matrices
   - Interstitial contribution in plane-waves
   - Sphere contributions in local basis
   - Matching of local basis to plane-waves
3. The diagonalization of the matrix
   - Call to optimized math-libraries
4. Calculation of new density
   - Sphere and interstitial density
   - Mixing of in- and output density

FLAPW-basis set defined in two different regions in space:

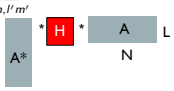
- Plane-waves
- Numerical (atom-like) functions in MT-sphere around each atom
- Matching across sphere boundary (A-coefficients)

$$\phi_j^{\mathrm{LAPW}} = \begin{cases} e^{i(\vec{k}+\vec{g}_j)\vec{r}} \\ \sum_{lm}^{L} A_{j,lm}\phi_{lm}^{\mathrm{atomic}}(\vec{r}) \end{cases}$$

Sphere contribution to Hamiltonian:

$$H_{i,j}^{\mathrm{MT}} = \sum_{\mathrm{atoms}} \sum_{lm,l'm'}^{L} A_{i,lm}^{*} H_{lm,l'm'}^{\mathrm{loc}} A_{j,l'm'}$$

Product of three matrices (Aspect ratio: N>>L)



Optimized Algorithm:

- Separation into spherical and non-spherical contribution

$$H_{i,j}^{\mathrm{MT}} = \sum_{\mathrm{atoms}} H_{i,j}^{\mathrm{loc;sph}} + H_{i,j}^{\mathrm{loc;non\text{-}sph}}$$

Spherical contribution:

- m,m'-sum can be performed analytically
- remaining single l-sum is fast

$$H_{i,j}^{\mathrm{loc;sph}} = \sum_{l} a_{i,l}^{*} H_{l,l}^{\mathrm{loc;sph}} a_{j,l} D_{i,j}$$

Non-spherical contribution:

- l,m sums can be restricted to smaller maximal L'

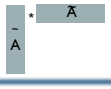
$$H_{i,j}^{\mathrm{loc;non\text{-}sph}} = \sum_{lm,l'm'}^{L'} A_{i,lm}^{*} H_{lm,l'm'}^{\mathrm{loc;non\text{-}sph}} A_{j,l'm'}$$

- Local Hamiltonian can be Cholesky-decomposed

$$= \sum_{lm,l'm',l''m''}^{L'} A_{i,lm}^{*} U_{lm,l''m''}^{*} U_{l'm',l''m''} A_{j,l'm'}$$

- Final operation is a rank-k update (zherk)

$$= \sum_{lm}^{L'} \tilde{A}_{i,lm}^{*} \tilde{A}_{j,lm}$$



Modernization of the Code

- Intensive support and discussions with HPC groups at JSC (Jülich Forschungszentrum) and IT Center (RWTH Aachen University)

JARA|HPC

- Identification of computational most relevant code segments

Code part	CPU time (24 core)	Speedup on 384 cores	Parallel efficiency
Setup,Potential	5%	3.1	20%
Matrix setup	45%	10	63%
Diagonalization	40%	9.0	56%
Charge density	8%	5.3	33%
Mixing	1%	2.2	14%
Total SCF run	**100%**	**7.8**	**49%**

256 atoms CuAg on CLAIX, scales up to 16 nodes (384 cores)

- Implementation of multi-level hybrid parallelism enabling the efficient use of multi-core and multi-node machines using MPI and OpenMP

Code Parts	Level of Parallelization		
	MPI	OpenMP	SIMD
Potential	- a lot of small subroutines - parallelization of loops	- loops	- compiler flags - BLAS calls

	k – points	Eigenvalue	
Matrix Setup	- independent eigenvalue problems for each k-point	- block distribution among the processes	- BLAS calls - compiler hints - compiler flags
Diagonalization		- interfaces to the external libraries: ELPA, ScaLAPACK, Elemental	
New Charge	- loop over reciprocal lattice vector $\vec{g}$		- loop over atoms - compiler flags
Mixing	- to be done		

Version 0.26 Version 0.27 MaX Release 2.0

- Interfacing of various high-performance libraries for matrix diagonalization (ELPA, ScaLAPACK, Magma, Elemental)
- Implementation of new algorithms suited for current and future computing architectures
- Removal of show-stoppers like excessive IO
- Code version to use CUDA programming for running on GPUs

NVIDIA CUDA

Improved FLEUR experience

Typical challenges for FLEUR users:

- All-electron method requires many parameters and has a complex input
- Input/Output difficult to understand and to modify
- Availability of documentation

Exemplary actions implemented:

- XML input/output
- More documentation on webpages/wiki

<xml /> doxygen MediaWiki

- Tutorials and hands-on sessions

Code refactoring and developers tools:

- New version with increased degree of modularization
- Removing of IO and implementing of structured parallel IO schemes where needed
- Automatic building and testing of executables in a continuous integration framework
- Introduction of modern Fortran programming feature

- Code available for collaborative development on institutes GitLab server:

https://iffgit.fz-juelich.de/fleur/fleur

GitLab

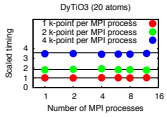
Acknowledgment

This work has been supported by the EU through the H2020-EINFRA-2015-1 project: GA 676598 and by a JARA-HPC seed-fund project.

Performance gains achieved

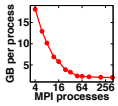
K-point MPI parallelization

- Independent eigenvalue problems
- Negligible communication
- Almost ideal scaling

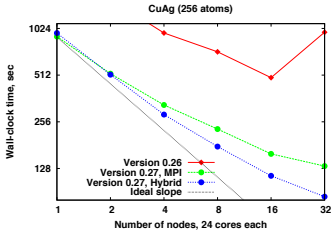


Eigenvalue parallelization

- Reduces the memory usage per MPI process
- Elimination of I/O improved performance considerably
- As #nodes grow, hybrid parallel scheme becomes more efficient



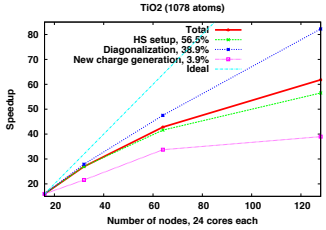
Comparison with old FLEUR version



@ CLAIX Intel Broadwell 15 min on 1 node

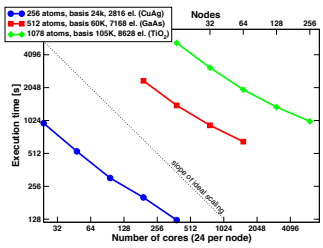
Large systems

- Simulation of unit cells with >1000 atoms is now possible

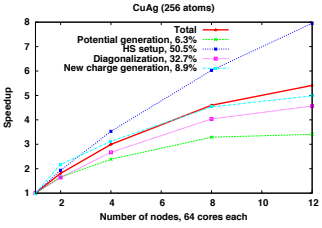


@ CLAIX Intel Broadwell 87 min on 16 nodes

Scaling for different systems/sizes



Accelerator Architectures



@ CLAIX Intel KNL 17 min on 1 node

Intel KNL:

- Comparable performance to Broadwell

NVIDIA GPU (K80):

- Only single node due to missing diagonalization library

Future Plans, Challenges

- Further porting to accelerators
- Further tuning of the KNL performance
- Redesign of matrix layout to enable easier implementation of new algorithms and easier tuning to new architectures