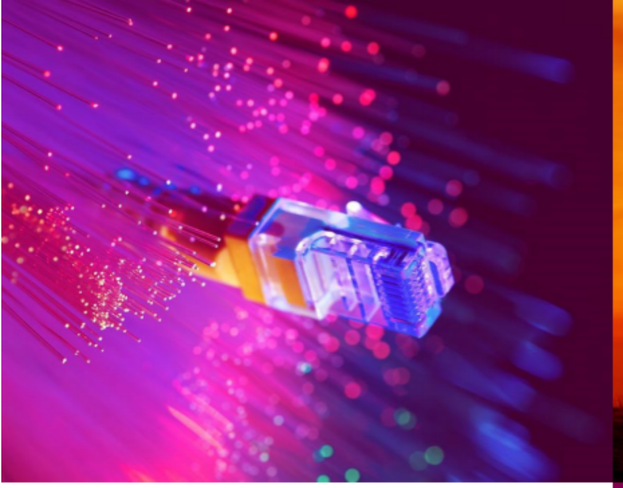


GPAW in ISC21 Student Cluster Competition

Jussi Enkovaara

Outline

- Overview of GPAW
- Parallelization in GPAW
- ISC21 inputs and tasks

About GPAW

- Open source software package for atomic scale quantum mechanical simulations
- Density-functional theory
- Supports multiple basis sets
- Implemented in Python and C programming languages
- Development started in early 2000 in Technical University of Denmark
- Currently, few hundred users and 10-20 active developers
- wiki.fysik.dtu.dk/gpaw

About me

- Ph.D. in Physics (Electronic structure simulations), Helsinki University of Technology (currently Aalto University) 2003
- Since 2005 worked at CSC - IT Center for Science as HPC specialist
- GPAW developer since 2005

Density-functional theory

- Many-body Schrödinger equation

$$H(r_1, r_2, \ldots, r_N)\Psi(r_1, r_2, \ldots, r_N) = E\Psi(r_1, r_2, \ldots, r_N)$$

$$H = \sum_i -\frac{\nabla_i^2}{2} + V_{ext}(r) + \frac{1}{2}\sum_{i \neq j} \frac{e^2}{|r_i - r_j|}$$

- Analytic solution for single electron
- Wavefunction Ψ is 3N dimensional
  - 10 electrons in 10x10x10 grid $\rightarrow 1000^{30}$ degrees of freedom
- Density-functional theory maps the problem into a set of single-particle equations

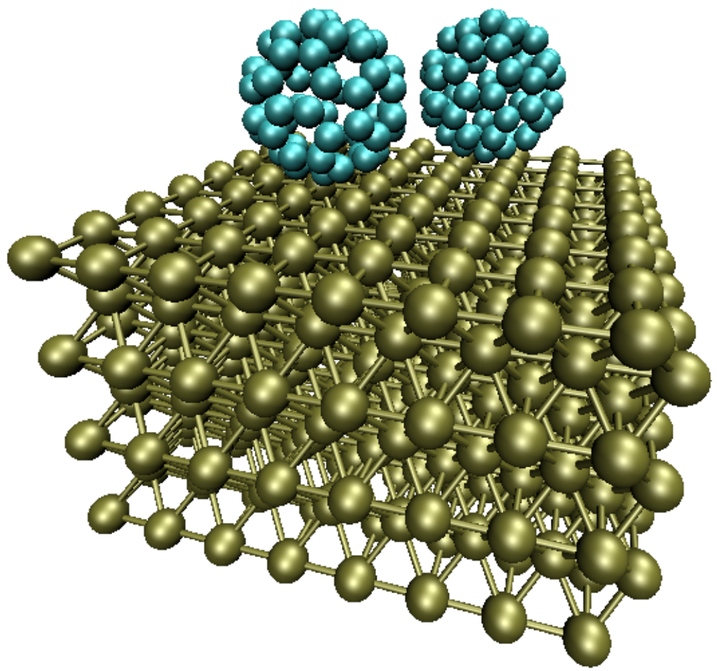
Kohn-Sham equations

$$\left(-\frac{\nabla^2}{2} + V_H((n(r)) + V_{xc}((n(r))\right)\psi_i(r) = \epsilon_i \psi(r)_i$$
$$n(r) = \sum_i |\psi_i(r)]^2$$

- Set of self-consistent equations:
  - Start with initial guess for density $n(r)$
  - Solve $\psi_i(r)$
  - Calculate new $n(r)$ and repeat until converged
- Physical approximations are contained in the exchange-correlation potential V_{xc}

Applications of density-functional theory

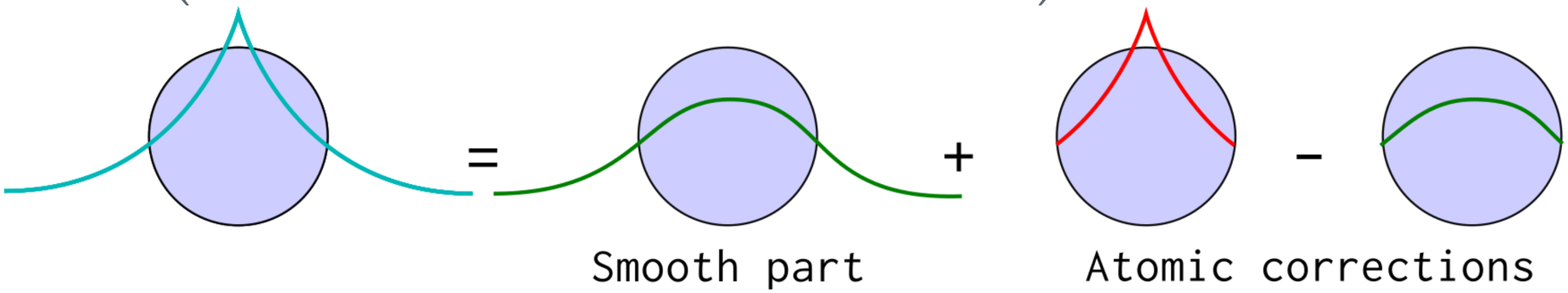
- Structure of matter (bond lengths, equilibrium crystal structures)
- Formation energies
- Ab-initio molecular dynamics
- Optical and magnetic properties
- Electronic structure
- …
- Major consumer of computational resources all over world



Projector-augmented wave method

- Projector-augmented wave method allows one to work with smoother pseudo-wave functions

$$\left(-\frac{\nabla^2}{2} + V_H + V_{xc} + \sum_a H^a < \tilde{p}^a|\right) \tilde{\psi}_i(r) = \epsilon_i \tilde{\psi}(r)_i$$

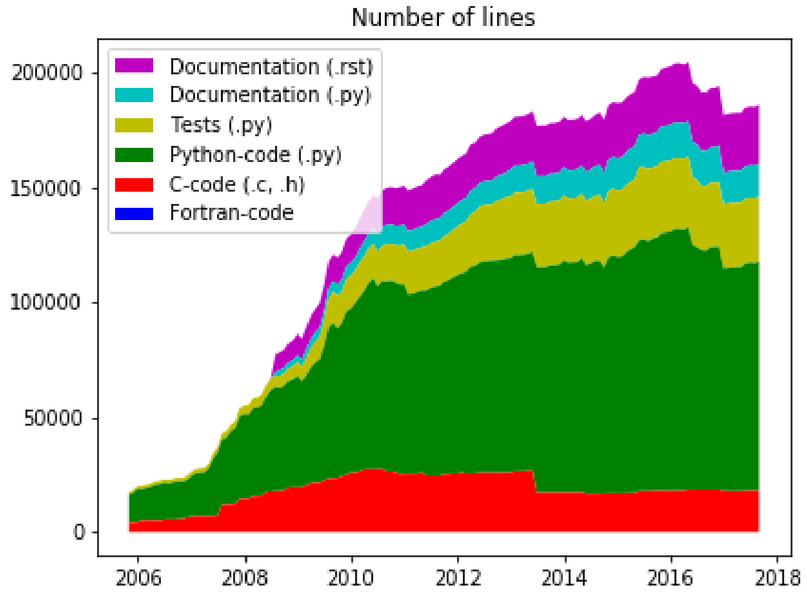


Basis sets in GPAW

- Uniform real-space grid, finite-difference stencil for ∇^2
  - Convergence parameter h, smaller more accurate
  - Good parallel scalability
- Plane waves
  - Convergence parameter plane wave cutoff, larger more accurate
  - Relies on Fast Fourier transforms
  - Only periodic boundary conditions
  - Parallel scalability limited by FFTs
- Atomic orbital basis set
  - Fast calculations, accuracy can be lower than with other basis sets
  - Systematic convergence difficult

Python implementation

- High-level algorithms are implemented in Python
- Input file is also a Python script utilizing Atomic Simulation Environment
- Computationally intensive parts implemented in C and in libraries
  - BLAS, FFTs, LAPACK, ScaLAPACK
- Typically, 90 - 95 % of total time spent in C or in libraries

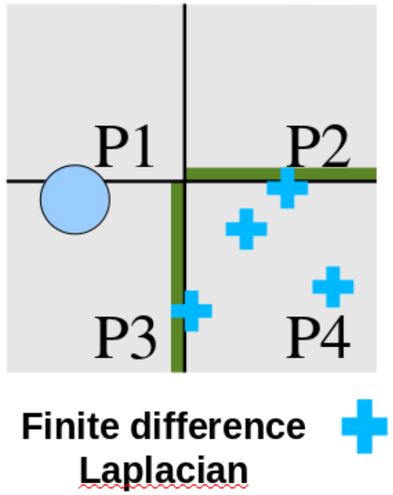


Parallelization in GPAW

- Main parallelization scheme MPI
  - MPI calls both from C and from Python
- Complementary OpenMP parallelization
  - Can be beneficial in supercomputers with many cores per node
  - Not fully optimized yet
  - Only real-space grids and atomic orbital basis
  - Multithreaded BLAS required for good performance
  - MPI library with MPI_THREAD_MULTIPLE support required

Parallelization in GPAW

- Parallelization over several degrees of freedoom
- k-points and spin
  - periodic and magnetic systems
  - nearly trivial parallelization
- Domain decomposition
  - real-space grids and atomic orbital basis
  - only local communication
- Parallelization over plane waves
  - all-to-all communication



Parallelization in GPAW

- Parallelization over several degrees of freedoom
- Parallelization over electronic states
  - can be beneficial when domain decomposition or parallelization over plane waves no longer scales
  - typically does not happen until using several hundreds of CPU cores
- Dense matrix diagonalizations with ScaLAPACK
  - with real-space and plane wave basis beneficially normally only for cases with over 1000 states
  - atomic orbital basis can benefit already with smaller systems

Installing GPAW

- If all non-Python requirements are met, GPAW can in principle be installed directly from PyPI (Python package index)
- In ISC21 SCC one should install version **21.1.0** from source:

```
git clone -b 21.1.0 https://gitlab.com/gpaw/gpaw.git
```

- Normally, one wants to set at minimum the BLAS library in `siteconfig.py`:

```
libraries = ['openblas']
library_dirs = ['/some/path/where/openblas/is/lib']
```

This will add `-L/some/path/where/openblas/is/lib -lopenblas` to link line when building GPAW

Installing GPAW

- By default, `mpicc` and options used for the Python interpreter are used
- Another compiler and additional flags can be set also in `siteconfig.py`
- See ISC21 SCC wiki or GPAW wiki for more details.

- Once installation is complete and `PATH` *etc.* are set, PAW datasets can be installed as

```
gpaw install-data <dir>
```

- Simple test calculation can then be performed with

```
gpaw test
```

- GPAW contains also a more extensive test set when developing code, see GPAW wiki for details

Running GPAW

- GPAW input files are Python scripts
  - Complex workflows can be programmed in the input file itself
- Syntactic correctness of input file and default parallelization settings with **N** processes can be checked with a *dry-run*

```
gpaw python --dry-run=N input.py
```

  - Note that output file defined in the input will be overwritten
- The way to start parallel calculations depends on the underlying batch job system and MPI installation (`mpiexec`, `srun`, ...), e.g. with `mpiexec`

```
# set PATH, PYTHONUSERBASE or PYTHONPATH etc.
mpiexec -n 40 gpaw python input.py
```

A look into GPAW input

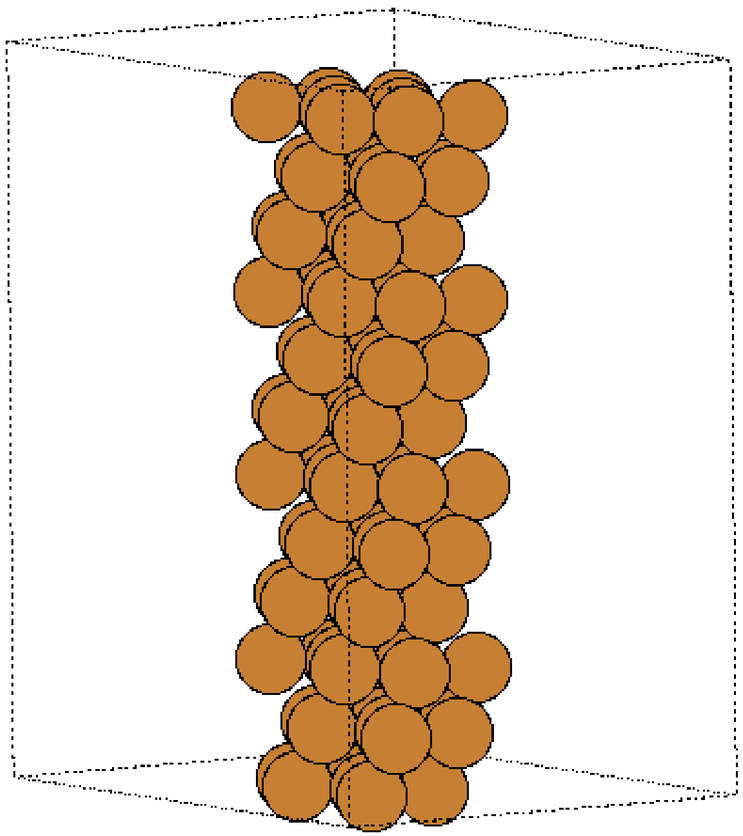
```
from ase.build import bulk        # Atomic simulation env tools
from gpaw import GPAW
from gpaw.mpi import world        # Information about parallelization

atoms = bulk('Si', cubic=True)

calc = GPAW(h=0.2,                # Accuracy of real space grid
            kpts=(3,3,3),         # K-point mesh (only with periodic systems
            xc='PBE',             # Exchange-correlation approximation
            txt=outfile,
           )

atoms.set_calculator(calc)
e = atoms.get_potential_energy()
if world.rank == 0:
    print("Energy:", e)
```

A look into GPAW output

```
...
Total number of cores used: 16
Parallelization over k-points: 4
Domain decomposition: 2 x 2 x 1
...
iter:   1  09:17:27                  -43.220046    1
iter:   2  09:17:28  +0.03  -0.82   -43.352201    1
iter:   3  09:17:29  -0.37  -0.82   -43.554556    1
...
Timing:                               incl.     excl.
------------------------------------------------------------
Hamiltonian:                           0.279     0.000   0.0% |
...
SCF-cycle:                            30.285     1.049   3.2% ||
 Davidson:                            16.935     9.209  27.7% |----------|
  Apply hamiltonian:                   1.217     1.217   3.7% ||
  Subspace diag:                       1.730     0.013   0.0% |
   calc_h_matrix:                      1.022     0.118   0.4% |
```

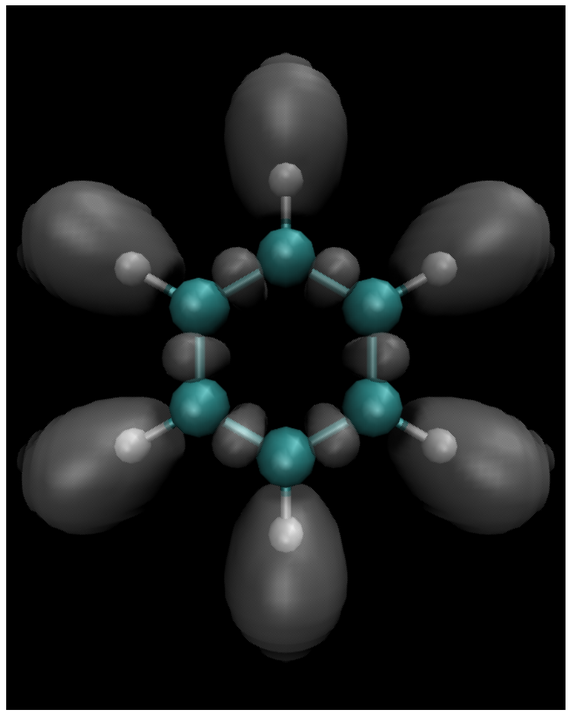
Tasks in competition: building and running

- Build GPAW in the two clusters
- Investigate and discuss the scalability in the two clusters
- Input case `copper.py`
  - Copper filament, periodic in z-direction
  - Real-space basis, k-points in z-dimension
  - Limited number of self-consistent iterations
- No modifications to the input



Tasks in competition: visualization

- Electron localization function (ELF) is a measure of the likelihood of finding an electron in the neighborhood space of a reference electron
- Can be used in interpreting and visualizing bonding
- Provided input `nanoribbon.py` writes out the 3D ELF (together with the atomic positions) of graphene nanoribbon with Au adsorbate
- Make a visualization of ELF



Tasks in competition: profiling

- Use IPM profiler to profile the input `copper.py` over 4 node run.
- (There is also input `copper-profile.py` which uses Python standard profiler and writes information into separate file for each MPI task. The profiles can be investigated with tools in Python standard library)

Tasks in competition: performance tuning

- Try to maximize the performance of `copper.py`
- You can try different compilers, compiler options, libraries
- Any modifications to source code are allowed (as long as the accuracy check in the input passes)
  - modifications need to be made available
- Non-default parallelization options are allowed, i.e. use of OpenMP threading and adding `parallel` keyword into input, e.g.

```
args = {'h': h,
        ...
        'txt': txt,
        'parallel' : {'band' : 2}
       }
```

Bonus task: bug fix for scalapack diagonalization

- There is a bug in GPAW's Scalapack functionality: https://gitlab.com/gpaw/gpaw/-/issues/269
- Try to fix the bug
- Note that GPAW needs to be built with ScaLAPACK support for this task

Coding challenge

- GPAW is used also in Coding Challenge for analyzing `MPI_Alltoallv` patterns
- Input file `si-divacancy.py`
  - Divacancy in Si
  - Plane wave basis

Questions ?