Time-dependent density-functional theory with massively parallel computers

Jussi Enkovaara

CSC – IT Center for Science, Finland

Outline

- Overview of the GPAW software package
- Parallelization for time-dependent density-functional theory
- Python implementation
- Results
- Summary

GPAW

- Quantum mechanical calculations of nanosystems
  - (Time-dependent) density-functional theory
  - System sizes up to few thousands of atoms
- Projector augmented wave method on
  - uniform real-space grids, atomic orbital basis, plane waves
- Open source software licensed under GPL
  - 20-30 developers in Europe and in USA

wiki.fysik.dtu.dk/gpaw

J. J. Mortensen et al., Phys. Rev. B 71, 035109 (2005)

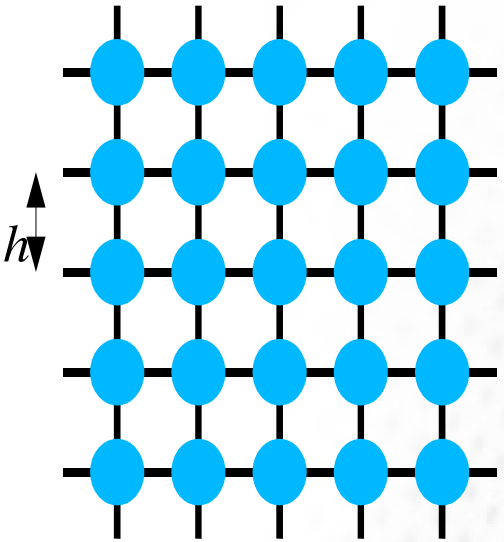
J. Enkovaara et al., J. Phys. Condens. Matter 22, 253202 (2010)

GPAW features

- Total energies, forces, structural optimization
  - analysis of electronic structure
- Excited states, optical spectra
  - Non-adiabatic electron-ion dynamics
- Wide range of XC-potentials (thanks to libxc!)
  - LDAs, GGAs, meta-GGAs, hybrids, DFT+U, vdW, RPA
- Electron transport
- GW-approximation, Bethe-Salpeter equation
- ...

Real-space grids

- Wave functions, electron densities, and potentials are represented on grids.
- Single parameter, grid spacing h



- Accuracy of calculation can be improved systematically
- Derivatives by finite differences

Time-dependent DFT

- Generalization of density-functional theory to time-dependent cases
- Runge-Gross theorem PRL 52 (1984)
  - one-to-one mapping between the time-dependent potential and the density
- Excited state properties
  - excitation energies, optical spectra, ...
- Can be implemented in real-time and linear response forms

Linear response TD-DFT in finite systems

- Excitation energies can be calculated from eigenvalue equation:

$$\mathbf{\Omega} F_I = \omega_I^2 F_I,$$

where

$$\Omega_{ij\sigma,kl\tau} = \delta_{ik}\delta_{jl}\delta_{\sigma\tau}\varepsilon_{ij\sigma}^2 + 2\sqrt{\varepsilon_{ij\sigma}\varepsilon_{kl\tau}}K_{ij\sigma,kl\tau},$$

$$\varepsilon_{ij} = \varepsilon_i - \varepsilon_j$$

with the coupling kernel

$$K_{ij\sigma,kl\tau} = \int dr_1 dr_2 n^*_{ij\sigma}(r_1) \left[\frac{1}{|r_1 - r_2|} + f_{xc}(r_1, r_2)\right] n_{kl\tau}(r_2)$$

$$n_{ij\sigma}(r) = \psi_{i\sigma}(r)^* \psi_{j\sigma}(r)$$

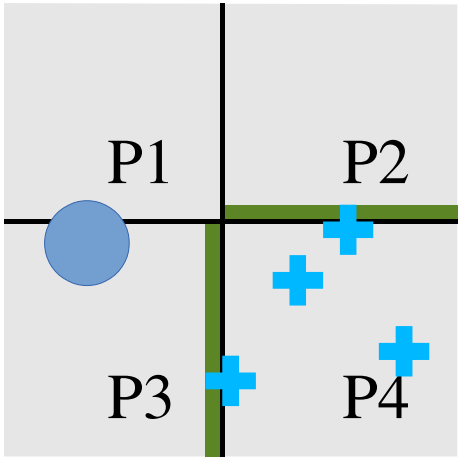
- i and j indexes go through occupied and unoccupied states, respectively

Parallelization over real-space grid

Coulombic part: $-\nabla^2 v_{kl}^K(r_1) = n_{kl}(r_1)$

XC part: $v_{kl}^{XC}(r_1) = \int dr_2 n_{kl}(r_2) f_{xc}(r_2, r_1)$

- Domain decomposition



Finite difference Laplacian

PAW augmentation sphere

- Only local communication
- Good parallel scalability down to domain sizes ~16 x 16 x 16

Parallelization over electron-hole pairs

- Casida equation in linear response TD-DFT:

$$\mathbf{\Omega} F_I = \omega_I^2 F_I,$$

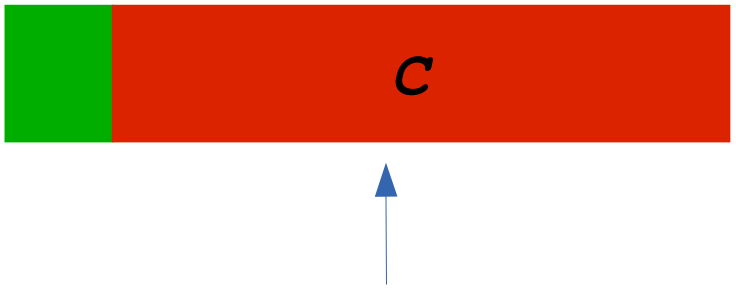
$$\Omega_{ij\sigma,kl\tau} = \delta_{ik}\delta_{jl}\delta_{\sigma\tau}\varepsilon_{ij\sigma}^2 + 2\sqrt{\varepsilon_{ij\sigma}\varepsilon_{kl\tau}}K_{ij\sigma,kl\tau}$$

- Matrix elements can be calculated independently
- Nearly trivial parallelization over electron-hole pairs ij
- Good scalability down to ~20 electron-hole pairs per MPI task

Python based implementation

Lines of code:

Execution time:



BLAS, LAPACK, MPI, NumPy

- Python (+ NumPy)
  - Fast development
  - Slow execution
  - High level algorithms
- C
  - Fast execution
  - Slow development
  - Main numerical kernels

Python based implementation

- Snippet from Poisson solver

```
# Call to C-functions
relax(phi, rho)
laplacian.apply(phi, residual)
# Python (NumPy)
residual -= rho
```

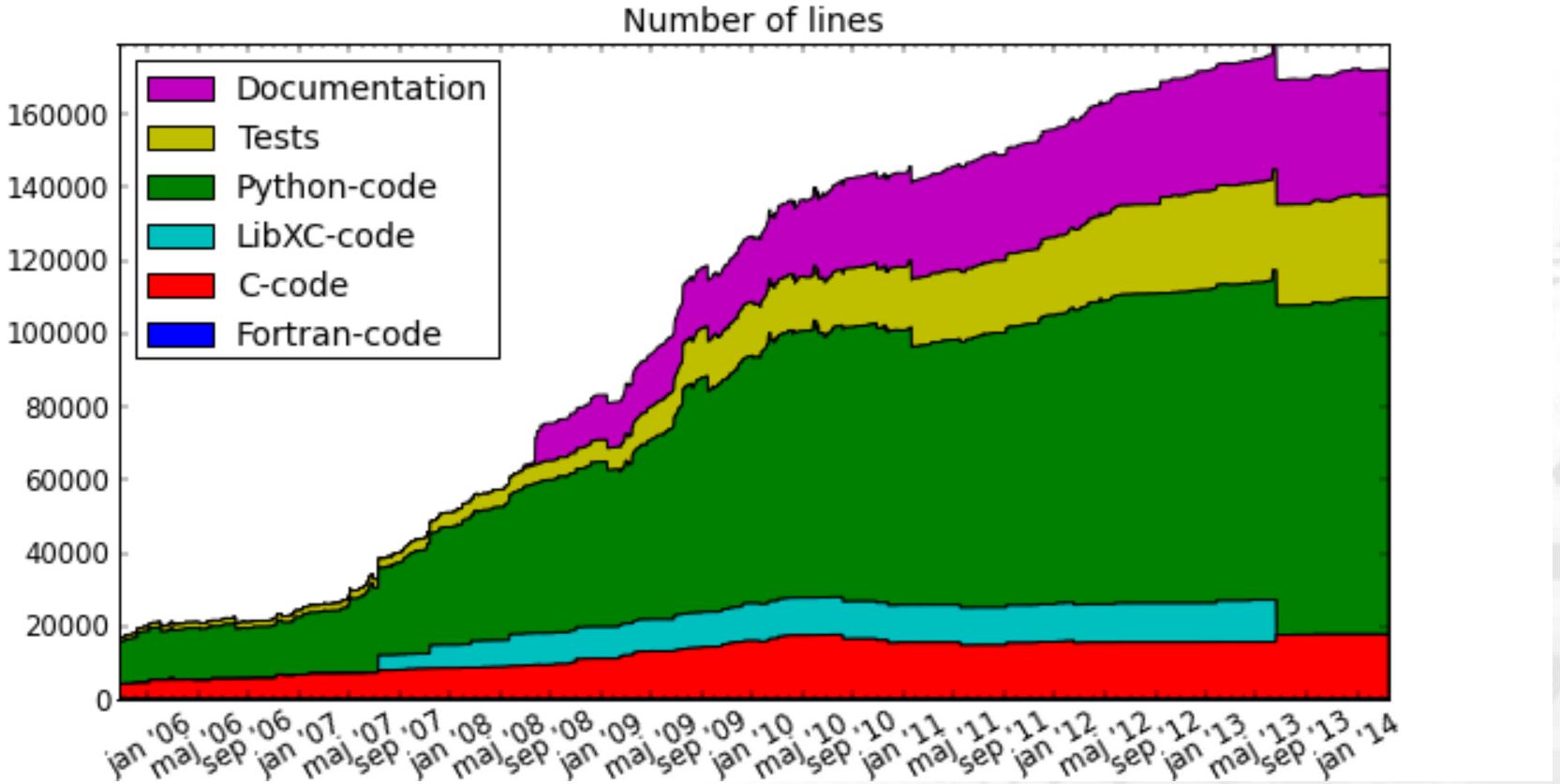
Parallel programming in GPAW

- Message passing with MPI
- Custom Python interface to MPI
- MPI calls both from Python and from C

```
# MPI calls within the apply C-function
laplacian.apply(phi, residual)
# Python interface to MPI_Reduce
error = gd.comm.sum(np.dot(residual,residual))
```

- Experimental:
  - MPI + OpenMP, threading within C-extensions
  - CUDA kernels

Python based implementation



Time line of GPAW's codebase

Python initialization

- **import** statements in Python trigger lots of small-file I/O
  - stat, fstat, fopen, fgetc, ...
  - dynamic loading of shared libraries
- In parallel calculations all processes perform the same I/O
- Introduces severe bottleneck with large number (> 1000-2000) of processes
  - Initialization can take more than 30 minutes !

Scalable Python interpreter

- Wrappers for C-stdio routines

```
int __wrap_fgetc ( FILE * fp )
{
  int x;
  if (rank == MASTER )
    {
      x = getc(fp);
      MPI_Bcast(&x, 1, MPI_INT, MASTER, io_comm);
    }
  else
      MPI_Bcast(&x, 1, MPI_INT, MASTER, io_comm);

  return x;
}
```

- Only single process performs I/O

Scalable Python interpreter

- Enable wrappers by including a special header in import related Python source files

parallel_stdio.h

```
#define fopen  __wrap_fopen
#define fclose  __wrap_fclose
...
```

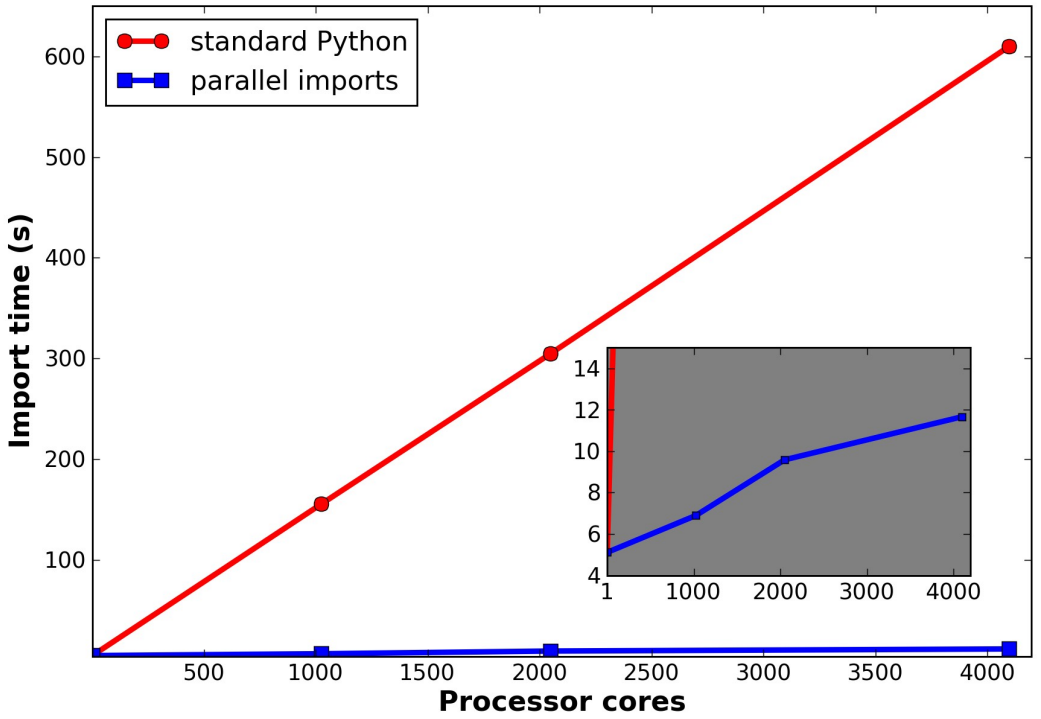
import.c

```
...
#include "importdl.h"
#ifdef ENABLE_MPI
#include "parallel_stdio.h"
#endif
...
```

gitorious.org/scalable-python

Original idea: **D. M. Beazley *et al*. Feeding a large-scale physics application to python, in: 6th International Python Conference, USENIX, 1997**

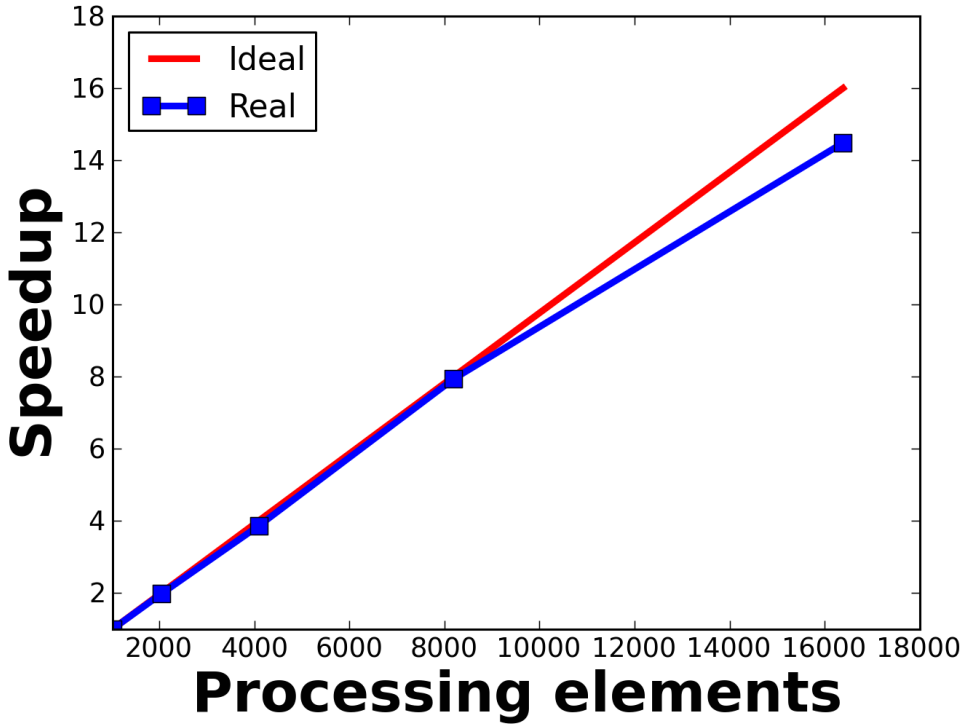
Scalable Python performance



Cray XE6 Hermit, HLRS, Germany

- Significant improvement in startup time
- Dynamic libraries remain problem

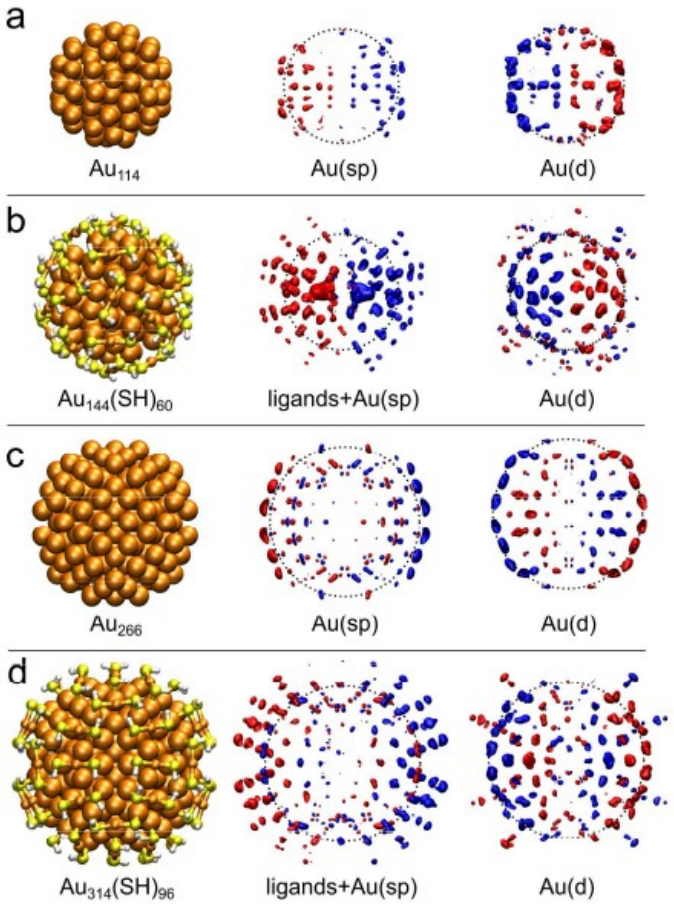
Parallel scalability of full calculation

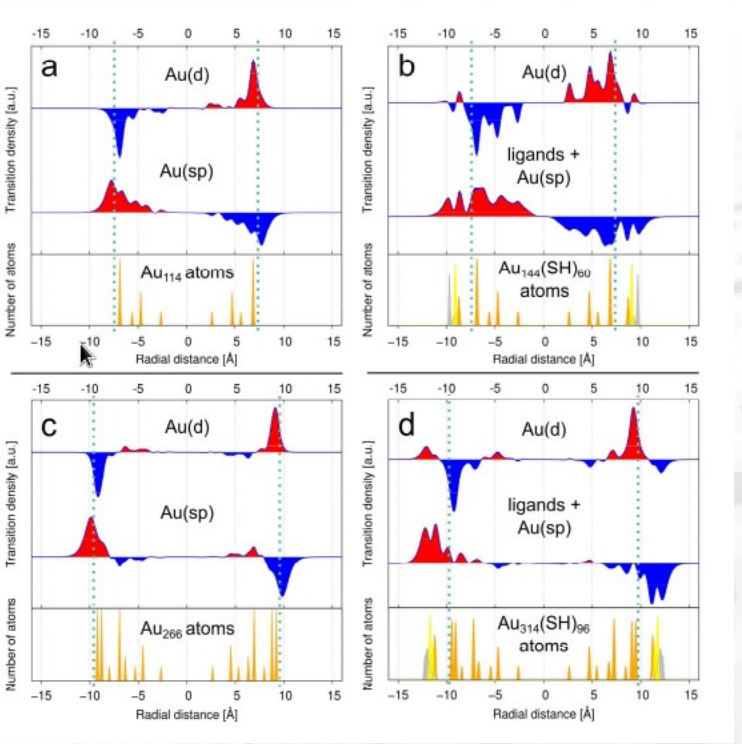


- Linear-response TD-DFT
  - Au38(SCH3)24 cluster ~160 atoms
  - ~680 electronic-hole pairs
  - Cray XE6 Hermit, HLRS, Germany

Example case

- Plasmonic optical spectra in Au clusters





- Production calculations with ~8000 cores

S. Malola, L. Lehtovaara, JE, H. Häkkinen, ACS Nano (2013)

Summary

- Linear response time-dependent density-functional theory offers good parallelization prospects
- Large scale parallel calculations are feasible with Python based software
- Python import mechanism is challenging in massively parallel scale

Acknowledgements

- Partnership for Advanced Computing in Europe (PRACE), EU FP7 programme
- Finnish Technology Agency, MASI program
- Argonne Leadership Computing Facility, US DoE
- Whole GPAW development team
  - J.J., Marcin, Ask, Nick, Christian, Carsten, Lauri, Michael, Lara, Mikael, Olga, Mikkel, Jun, Thomas, Jess, Mathias, Ari, Samuli, David, Karsten, Jens, Kristian, Jacob, Hannu, Tapio, Martti, Risto, ...

Acknowledgements