Massively parallel electronic structure calculations with Python software

Jussi Enkovaara
Software Engineering
CSC – the finnish IT center for science
GPAW

- Software package for electronic structure calculations within the density-functional theory
- Python + C programming languages
- Massively parallelized
- Open source software licensed under GPL

www.csc.fi/gpaw

wiki.fysik.dtu.dk/gpaw
Collaboration

Tech. Univ. of Denmark
- J. J. Mortensen
- M. Dulak
- C. Rostgaard
- A. Larsen
- K. Jacobsen

Helsinki Univ. of Tech.
- L. Lehtovaara
- M. Puska
- R. Nieminen
- T. Eirola

Tampere Univ. of Tech.
- J. Ojanen
- M. Kuisma
- T. Rantala

Jyväskylä University
- M. Walter
- O. Lopez
- H. Häkkinen

Åbo Akademi
- J. Stenlund
- J. Westerholm

- Funding from TEKES
Outline

- Density functional theory
- Uniform real-space grids
- Parallelization
  - domain decomposition
  - Python + C implementation
  - results about parallel scaling
  - future prospects
- Summary
Density functional theory

- Calculation of material properties from basic quantum mechanics
- First-principles calculations, atomic numbers are the only input parameters
- Currently, maximum system sizes are typically few hundred atoms or 2-3 nm
- Numerically intensive simulations, large consumer of supercomputing resources
Density functional theory

- Kohn-Sham equations in the projector augmented wave method
  \[
  \left(-\frac{\nabla^2}{2} + V_{eff}(r)\right)\psi_i(r) = e_i\psi_i(r)
  \]
  \[
  n(r) = \sum_i |\psi_i(r)|^2
  \]
  \[
  V_{eff}(r) = V_{ext}(r) + V_{Coul}(r) + V_{xc}(n(r))
  \]
  \[
  \nabla^2 V_{Coul}(r) = 4\pi n(r)
  \]
  \[
  V_{eff}(r) = \tilde{V}_{eff}(r) + \sum_a V_{nl}^a(r, r')
  \]

- Self-consistent set of single particle equations for N electrons
- Non-locality of effective potential is limited to atom-centered augmentation spheres
Real space grids

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

- Accuracy of calculation can be improved systematically by decreasing the grid spacing
Finite differences

- Both the Poisson equation and kinetic energy contain the Laplacian which can be approximated with finite differences

\[
\frac{\partial^2 \psi(x_i)}{\partial x^2} = \sum_{n=-N}^{N} C_n \psi(x_i + nh) + O(h^{2N+2})
\]

- Accuracy depends on the order of the stencil N
- Sparse matrix, storage not needed
- Cost in operating to wave function is proportional to the number of grid points
Multigrid method

- General framework for solving differential equations using a hierarchy of discretizations

- Recursive V-cycle

- Transform the original equation to a coarser discretization
  - restriction operation

- Correct the solution with results from coarser level
  - interpolation operation
Domain decomposition

- Domain decomposition: real-space grid is divided to different processors
- Communication is needed:
  - Laplacian, nearly local
  - restriction and interpolation in multigrid, nearly local
  - integrations over augmentation spheres
- Total amount of communication is small
Python

- Modern, object-oriented general purpose programming language
- Rapid development
- Interpreted language
  - possible to combine with C and Fortran subroutines for time critical parts
- Installation can be intricate in special operating systems
  - Catamount, CLE
  - BlueGene
- Debugging and profiling tools are often only for C or Fortran programs

**Comparison: Lines of code:**

<table>
<thead>
<tr>
<th>Python</th>
<th>C</th>
</tr>
</thead>
</table>

**Comparison: Execution time:**

| C |

- BLAS, LAPACK, MPI, numpy
Python overhead

- Execution profile of serial calculation (bulk Si with 8 atoms)

<table>
<thead>
<tr>
<th>ncalls</th>
<th>tottime</th>
<th>percall</th>
<th>cumtime</th>
<th>percall</th>
<th>filename:lineno(function)</th>
</tr>
</thead>
<tbody>
<tr>
<td>76446</td>
<td>28.000</td>
<td>0.000</td>
<td>28.000</td>
<td>0.000</td>
<td>:0(add)</td>
</tr>
<tr>
<td>75440</td>
<td>26.450</td>
<td>0.000</td>
<td>26.450</td>
<td>0.000</td>
<td>:0(integrate)</td>
</tr>
<tr>
<td>26561</td>
<td>13.316</td>
<td>0.001</td>
<td>13.316</td>
<td>0.001</td>
<td>:0(apply)</td>
</tr>
<tr>
<td>1556</td>
<td>12.419</td>
<td>0.008</td>
<td>12.419</td>
<td>0.008</td>
<td>:0(gemm)</td>
</tr>
<tr>
<td>80</td>
<td>11.842</td>
<td>0.148</td>
<td>11.842</td>
<td>0.148</td>
<td>:0(r2k)</td>
</tr>
<tr>
<td>84</td>
<td>4.470</td>
<td>0.053</td>
<td>4.470</td>
<td>0.053</td>
<td>:0(rk)</td>
</tr>
<tr>
<td>3040</td>
<td>2.957</td>
<td>0.001</td>
<td>10.920</td>
<td>0.004</td>
<td>preconditioner.py:29(<strong>call</strong>)</td>
</tr>
<tr>
<td>14130</td>
<td>2.815</td>
<td>0.000</td>
<td>2.815</td>
<td>0.000</td>
<td>:0(calculate_spinpaired)</td>
</tr>
<tr>
<td>76</td>
<td>2.553</td>
<td>0.034</td>
<td>104.253</td>
<td>1.372</td>
<td>rmm_diis.py:39(iterate_one_k_point)</td>
</tr>
<tr>
<td>103142</td>
<td>2.390</td>
<td>0.000</td>
<td>2.390</td>
<td>0.000</td>
<td>:0(matrixproduct)</td>
</tr>
</tbody>
</table>

- 88 % of total time is spent in C-routines
- 69 % of total time is spent in BLAS
- In parallel calculations also Python parts are run on parallel
Implementation details

- **Message passing interface (MPI)**
- **Finite difference Laplacian, restriction and interpolation operators are implemented in C**
  - MPI-calls directly from C
- **Higher level algorithms are implemented in Python**
  - Python interfaces to BLAS and LAPACK
  - Python interfaces to MPI functions

```python
# Calculate the residual of pR_G, dR_G = (H - e S) pR_G
hamiltonian.apply(pR_G, dR_G, kpt)
overlap.apply(pR_G, self.work[1], kpt)
axpy(-kpt.eps_n[n], self.work[1], dR_G)

RdR = self.comm.sum(real(npy.vdot(R_G, dR_G)))
```

Python code sniplet from iterative eigensolver
Parallel scaling of multigrid operations

- Finite difference Laplacian, restriction and interpolation applied to wave functions
- Poisson equation (double grid)
Parallel scaling of whole calculation

- Realistic test and production systems

- 256 water molecules
  768 atoms, 1024 bands, 96 x 96 x 96 grid

- Au-(SMe) cluster
  327 atoms, 850 bands, 160 x 160 x 160 grid
Bottlenecks in parallel scalability

- **Load balancing**
  - atomic spheres are not necessarily divided evenly to domains

- **Latency**
  - Currently, only single wave function is communicated at time, many small messages
  - minimum domain dimension 10-20

- **Serial \(O(N^3)\) parts**
  - insignificant in small to medium size systems
  - starts to dominate in large systems
Additional parallelization levels

- In (small) periodic systems parallelization over k-points
  - almost trivial parallelization
  - number of k-points decreases with increasing system size

- Parallelization over spin in magnetic systems
  - trivial parallelization
  - generally, doubles the scalability

- Parallelization over electronic states (work in progress)
  - In ground state calculations, orthogonalization requires all-to-all communication of wave functions
  - In time-dependent calculations orthogonalization is not needed, almost trivial parallelization
Summary

- GPAW is a program package for electronic structure calculations within density-functional theory
- Implementation in Python and C
- Domain decomposition scales to over 1000 cores
- Additional parallelization levels could extend the scalability to > 10 000 cores

www.csc.fi/gpaw

wiki.fysik.dtu.dk/gpaw