Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Improving the Scalability of the Overlapping Fragments Method Code

Nenad Vukmirović[a]*, Petar Jovanović[a], Marko Mladenović[a]

[a]*Scientific Computing Laboratory, Institute of Physics Belgrade, University of Belgrade, Pregrevica 118, 11080 Belgrade, Serbia*

Abstract

The overlapping methods code for electronic structure calculations of large organic systems was benchmarked and profiled. Based on the profiling results, several performance and scalability bottlenecks caused by communication were identified. The code was then refactored to improve its performance with respect to these identified weak points. The modified code extends the maximal system size where good scaling is observed on Curie machine from 4,000 atoms to 16,000 atoms. On the other hand, we found that on Hermit machine the original code already exhibits rather good scaling and consequently the modified code leads to minor improvements only.

1. Introduction

Overlapping fragments method (OFM) [1] is a technique for finding the energies and wave functions of electronic states in semiconducting materials and nanostructures. Within OFM, one solves an eigenvalue problem of the form

$$\left(-\frac{\hbar^2}{2m_0}\nabla^2 + v_I + v_{\mathrm{H}} + v_{xc}^{LDA}(\rho)\right)\psi_{\mathrm{i}}(\boldsymbol{r}) = \varepsilon_i\psi_{\mathrm{i}}(\boldsymbol{r}), \qquad (1)$$

where v_I is the electrical potential of atomic nuclei, $v_H(\boldsymbol{r}) = \frac{e}{4\pi\varepsilon_0}\int \frac{\rho(\mathbf{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|}\mathrm{d}^3\boldsymbol{r}'$ is the Hartree potential of electrons in the system, $\rho(\boldsymbol{r})$ is the electronic charge density and $v_{xc}^{LDA}(\rho)$ is a known function of ρ (the so called exchange-correlation potential within the local density approximation). $\psi_{\mathrm{i}}(\boldsymbol{r})$ is the wave function of electronic state i, while ε_i is the energy of that state.

There is a wealth of methods that can be used to numerically solve Eq. (1); one can represent the wave functions in real space or using the linear combination of plane waves or Gaussian-type orbitals. However, in each of these cases one ends up with a very large matrix that needs to be diagonalized. The main idea of the OFM is to divide the system into fragments and represent the wave functions as linear combinations of orbitals of these fragments:

$$\psi_i(\boldsymbol{r}) = \sum\alpha_k^{(i)}\phi_k(\boldsymbol{r}).$$

The matrix that has to be diagonalized is then of significantly smaller dimensions.

OFM was developed with a focus to understand the electronic states in semiconducting organic polymer materials. These materials are, to a large extent, disordered and there is a strong need for large supercell calculations that would provide reliable information about the degree of localization of electronic states, the

* Corresponding author. *E-mail address*: nenad.vukmirovic@ipb.ac.rs

density of states, and eventually the electronic transport properties. For such systems, atomic positions can be generated from classical molecular dynamics but the challenge remains to calculate the electronic states. The OFM excellently complements the charge patching method (CPM) [2] for the construction of electronic charge density in the material. OFM is well suited for parallelization as it is based on the division of the system into fragments.

In this paper, we present the work focused on improving the performance and scalability of the OFM code. In Section 2 we describe the original code. Section 3 is devoted to the profiling results and the description of the code refactoring performed to improve the code. Finally, in Section 4 we present the results on scaling and performance of the modified code.

2. The Overlapping Fragments Method Code

In this section, we describe the details of the original OFM code. The code was written in Fortran and parallelized using MPI (therefore the number of processes invoked in runtime is equal to the number of requested CPU cores). The inputs to the code are the positions of atoms in the system and the single-particle potential, while the output gives the transfer integrals $H_{mn} = \langle \phi_m | H | \phi_n \rangle$ - where H is the single-particle Hamiltonian given by the term in brackets in Eq. (1) - and the overlaps of fragment orbitals $S_{mn} = \langle m | n \rangle$.

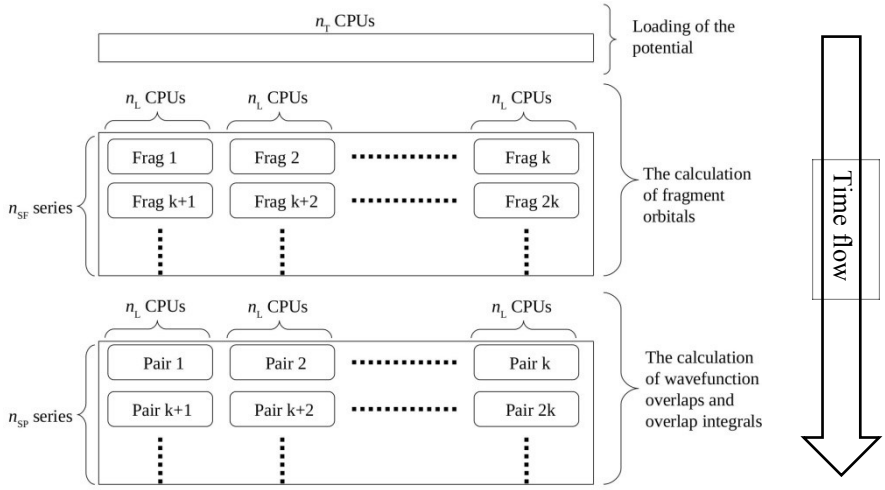

Figure 1: Temporal succession of OFM computation.

In the first part of the code the single-particle potential is loaded from disk (Figure 1). Since the single-particle potential cannot be stored in the memory of one node, it is divided into n_T parts of nearly equal size, each of which corresponds to one of the n_T processes (where n_T is the total number of processes). The master process reads the data from the input file and sequentially sends the data to each of the other processes.

In the second part of the code, the fragment orbitals are calculated. n_L processes are allocated to each fragment, where n_L is some small number (typically $n_L = 8$ or 16). The orbitals of the fragment are then calculated by solving the eigenvalue problem given in Eq. (1) using the combination of the CPM and the ESCAN code [3]. At this stage, the terms $H|\phi_n\rangle$ are also calculated, which will be later required for the evaluation of H_{mn}. To achieve this, the required real space grid values of the potential have to be sent to n_L processes allocated for fragment n. If n_F is the number of fragments in the system and $k = \lceil n_T/n_L \rceil$, the fragments are divided into $n_{SF} = \lceil n_F/k \rceil$ series. The calculations on the fragments from the same series are performed in parallel, where each fragment uses its n_L allocated processes. Such calculations are then repeated for all n_{SF} series of fragments.

In the third part of the code, the calculation of H_{mn} and S_{mn} terms is performed. Let n_P be the number of pairs for which H_{mn} and S_{mn} have to be calculated. For each pair, n_L processes are allocated where the calculation is performed. The pairs are divided into $n_{SP} = \lceil n_P/k \rceil$ series in a similar manner as for fragments in the second part of the code. To perform the calculation of H_{mn} and S_{mn} for the fragment pair (m,n) on its allocated n_L processes, these processes need to receive the orbitals ϕ_m, ϕ_n, $H|\phi_m\rangle$ and $H|\phi_n\rangle$ which are assigned to different groups of processes. In a similar manner as for fragments, the calculations for pairs from the same series are performed in parallel and then sequentially repeated for all n_{SP} series of pairs.

3. Profiling and Refactoring

Profiling was done on the thin nodes of Curie [4], using the TAU profiler [5]. The thin node partition consists of 5040 nodes, each having two eight-core Intel Xeon SandyBridge processors, with 64 GB of RAM per node and Infiniband QDR Full Fat Tree interconnect. It was found that the frequency and volume of communication between processes had good layout corresponding to their locality on nodes, as shown in Figure 2.

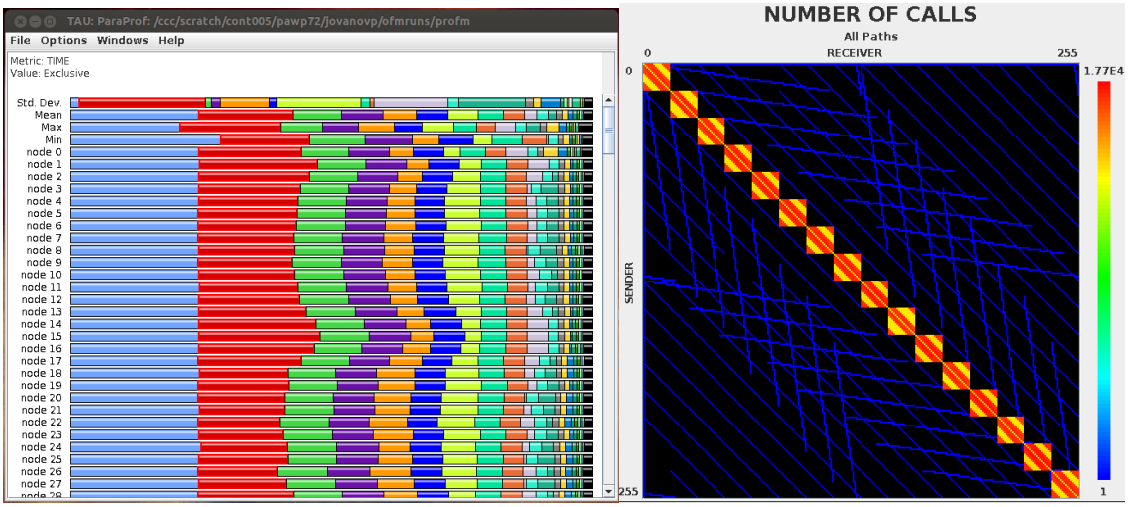

Figure 2: Profiling results for the smallest system investigated (consisting of 1,008 atoms). Bars on the left show exclusive time spent in functions for each process. The red bars represent cumulative time spent in all MPI_Barrier calls. The graph on the right shows communication matrix between each pair of processes.

The hotspots identified by the profiler were for the most part computations, but there was a significant time spent in MPI_Barrier calls. A similar pattern was observed for larger systems. The barriers are scattered throughout the code and it has proven difficult to remove them and maintain correctness, so this was left for some future development.

Despite the fact that the communication pattern in Figure 2 looks good and that one may conclude that the communication does not represent a bottleneck, a detailed analysis of the execution time of different parts of the code indicated that significant time is spent in the parts of the code that include communication (for example, the total wall time needed to perform one series in third part of the code increases from 1 second to around 30 seconds when the system size is increased from 1,000 to 64,000 atoms, while this time would not change for infinitely fast communication). Therefore, we focused on improving the communication in these parts of the code.

The performance and the scalability of the original code were improved by making modifications along the following lines:

- In the original code, the input potential was read from an input file and it was then distributed among all n_T processes in such a way that each process contains only one part of the potential. We have reorganized the way in which the potential is stored in memory for the reasons that will be described in the next bullet point. Instead of having a single copy of the potential divided among all n_T processes, we use several copies of the potential divided among a group of $n_V < n_T$ processes (when $n_L = 8$ we use $n_V = 32$). The motivation for this was to reduce the communication pressure for each of the n_T processes and improve the locality by replicating potential data to more subgroups of n_V processes in the second part of the code. Consequently, the first part of the original code was modified as follows. The input potential is read from disk and it is sent only to one group of n_V processes. It is subsequently broadcast to all other groups of n_V processes.

- In the second part of the code, each fragment that is stored on a small number of n_L processes needs only a part of the input potential. In the original code, the potential was divided among all n_T processes and significant time was needed to communicate it to a given fragment since the fragment needed to receive the data from all processes. With the reorganized way of storing the potential, each fragment needs to receive the data from a small local group of n_V processes only, which significantly speeds up the required communication. This improvement can be quantified from the results presented in Figure 4(a). Therefore, we have modified the second part of the code by introducing the procedure for

communicating a part of the local copy of the potential to a given fragment and removing the previous procedure for communication of the potential.

- In the third part of the code, one needs to calculate the overlap between wave functions of different fragments. To achieve this task one needs to bring these wave functions to the same group of n_L processes. In the original code, this communication was performed using a combination of send and receive commands and appeared to be inefficient. We have changed this part of the code by implementing the communication using mpi_alltoallv command which significantly reduced the time for this communication. This improvement can be quantified from the results presented in Figure 4(c).

4. Results

The original OFM code was initially benchmarked by performing a weak scaling test on Curie machine. The runs were performed for the poly(3-hexylthiophene) (P3HT) polymer system. The smallest system consists of four P3HT chains, each 10 thiophene rings long and contains 1,008 atoms altogether. The run for that system was performed with 256 CPU cores. The runs for larger systems were performed up to the system with 64,512 atoms, while the number of CPU cores used was increased proportionally to the number of atoms up to 16,384 cores. Figure 3 shows the dependence of CPU time (defined as the wall time multiplied by the number of CPU cores used) on the number of atoms.

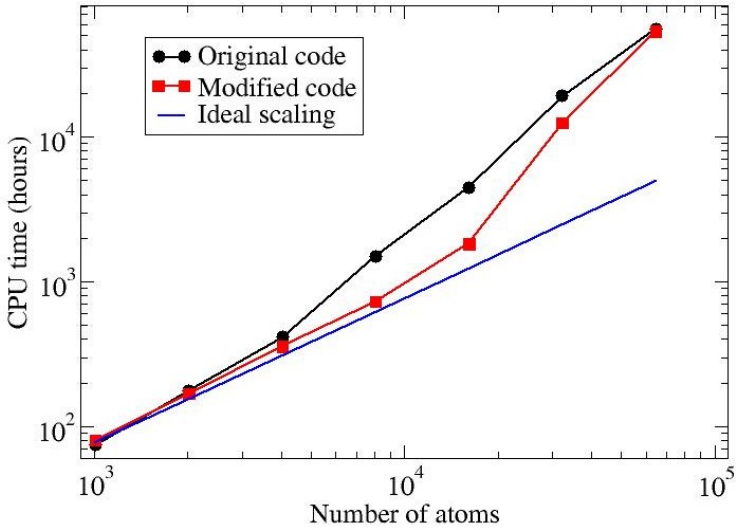

Figure 3: The dependence of CPU time on the number of atoms in the system for the simulations performed on Curie. The number of CPU cores used for the smallest system was 256 and was increased proportionally to the number of atoms for larger systems.

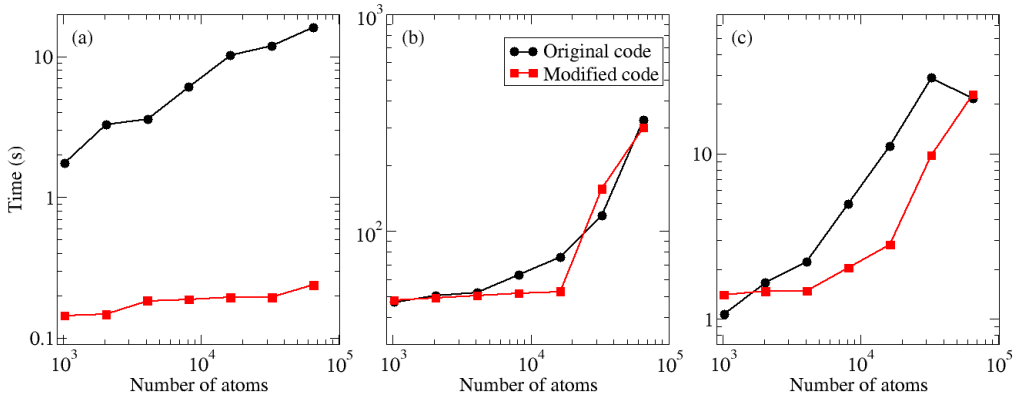

Figure 4: The dependence of wallclock time of different parts of the code on the number of atoms in the system for the simulations performed on Curie: (a) the time needed to communicate the potential in second part of the code; (b) the time for execution of one series (out of 4 series) in second part of the code; (c) the time for execution of one series in third part of the code (out of 44 series).

As seen from Figure 3 (black circles) the original code exhibited good scalability up to 4,000 atoms (1,000 cores). We have identified three weak points in the code where communication significantly slows down the code and we have refactored these parts of the code to improve its performance and scalability, as described in the previous section. Finally, we have performed the profiling of the modified code, which shows improvements in comparison to the original code. The performance of the code with the introduced improvements is shown in Figure 3 (red squares). These improvements extended the range of system sizes where scaling of the code is rather good up to 16,000 atoms (4,000 cores).

In Figure 4, we analyse the influence of modifications in parts of the code on overall code performance. The first part of the code has a negligible influence on the total execution time and is therefore not discussed. In Figure 4(a) we show the time needed to perform the communication of the potential in one series of the second part of the code. It can be seen from Figure 4(a) that this time is strongly reduced in the modified code by a factor between 10 and 70. In Figure 4(b), we show the total execution time of one series in the second part of the code (out of 4 series). Reducing the time to communicate potential reduces this time as well, however, for size beyond 20,000 atoms other reasons (which we have not identified) lead even to increase in this time. In Figure 4(c), we present the time needed to execute one series (out of 44 series) in the third part of the code. This time is significantly reduced in the modified code (by a factor ranging from 1.5 to 4 for test cases with 4,000 up to 16,000 atoms) and most significantly contributes to the reduction of execution time of the whole code.

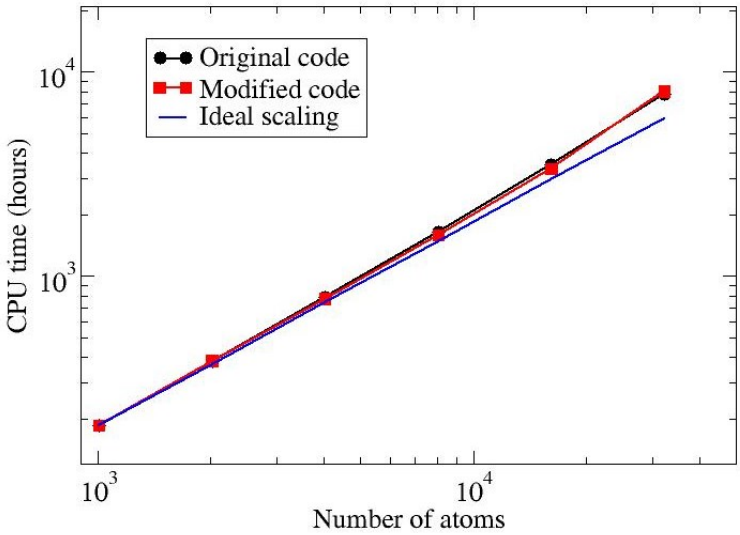

Figure 5: The dependence of CPU time on the number of atoms in the system for the simulations performed on Hermit. The number of CPU cores used for the smallest system was 256 and was increased proportionally to the number of atoms for larger systems.

The previously discussed runs were performed on the Curie system. To test the performance of the old and the new code on a different architecture, we have benchmarked the old and the new code on the Hermit system [6]. Hermit is a CRAY XE6 system that consists of 3552 nodes, each having two 16-core AMD Interlagos processors with 32 GB of RAM per node and CRAY Gemini node-node interconnect. The obtained CPU times are presented in Figure 5. Interestingly, one obtains a rather different code scaling behaviour on Hermit than on Curie. We find that the original code has rather good scaling performance on Hermit. Consequently, the improved code has only a slightly better performance.

Since we have identified that the communication affects the code performance on Curie, we believe that the origin of better scaling behaviour on Hermit in comparison to Curie is a different node interconnect with corresponding MPI libraries which leads to better communication in the case of OFM code.

5. Conclusion

In conclusion, we have refactored the OFM code in such a way that its scaling on the Curie machine is improved and we have established that the original code already has good scaling properties on the Hermit machine. Speedups of the modified code with respect to the original code obtained on Curie for systems ranging from 4,000 to 32,000 atoms take the value between 1.2 and 2.5.

6. References

[1] N. Vukmirović, L.-W. Wang, *J. Chem. Phys.* **2011**, 134, 094119.
[2] N. Vukmirović, L.-W. Wang, *J. Chem. Phys.* **2008**, 128, 121102.
[3] A. Canning, L.-W. Wang, A. Williamson, A. Zunger, *J. Comp. Phys.* **2000**, 160, 29-41.
[4] http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
[5] http://tau.uoregon.edu/
[6] http://www.hlrs.de/systems/platforms/cray-xe6-hermit/

7. Acknowledgements

This work was financially supported by the PRACE-3IP project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-312763.