Computing and Informatics, Vol. 30, 2011, 761–771

EFFECTIVE RESOURCE ALLOCATION IN PARALLEL QUANTUM-CHEMICAL CALCULATIONS

Grzegorz Mazur

Department of Computational Methods in Chemistry
Jagiellonian University
Ingardena 3, 30-060 Kraków, Poland
e-mail: `mazur@chemia.uj.edu.pl`

Marcin Makowski

Department of Theoretical Chemistry
Jagiellonian University
Ingardena 3, 30-060 Kraków, Poland
e-mail: `makowskm@chemia.uj.edu.pl`

Mateusz Brela

Faculty of Chemistry
Jagiellonian University
Ingardena 3, 30-060 Kraków, Poland
e-mail: `mbrela@student.chemia.uj.edu.pl`

Communicated by Jacek Kitowski

Abstract. Key factors affecting the parallel efficiency of archetypical quantum-chemical calculations are discussed. Effective load balancing schemes are proposed. Introduction of the memory affinity to the balancing process is shown to result in super-linear scaling.

Keywords: Computational chemistry, parallelization, optimization, super-linear scaling

1 INTRODUCTION

Significantly growing demand for higher quality computational chemistry results for large molecular systems can be recently observed. As a result, the chemistry community needs for computational power are constantly growing. While this trend is accompanied by quick development of high performance hardware solutions, the typical architecture of modern supercomputers and growing popularity of computer clusters require algorithmic changes in software to efficiently use the accessible resources.

The key issue is that the cluster architecture is inherently highly parallel. The same holds for virtually all modern supercomputers. This necessitates the switch from traditional sequential algorithms to those that are able to exploit the parallelism of the hardware architecture. Fortunately, most of the typical quantum chemistry problems are relatively easy to reformulate in such a way.

Most of the existing procedures used for commonly executed quantum-chemical calculations are parallelized within the Single Program Multiple Data (SPMD) model. This approach is in general not optimal from the efficiency point of view. It is, however, justified by its conceptual simplicity and near-optimal performance for typical quantum-chemical applications. The SPMD model requires a load-balancing algorithm to partition the problem into tasks performed by the computational nodes. The partitioning applies to all resources, which in case of typical quantum chemical calculations means the CPU time and memory.

In the paper we review archetypical quantum-chemical algorithms, analyzing how their key features affect their parallel performance. This allows us to formulate effective load-balancing schemes.

The proposed load-balancing procedures were implemented in development version of Niedoida [1], a general-purpose computational chemistry package built as a set of libraries implementing quantum-chemical and microelectrostatic calculations. Niedoida is a parallel program designed according to the SPMD model. Message Passing Interface (MPI) [2] is used as the low-level parallelization framework. The main reasons for using Niedoida in this work is that it provides a plug-in load-balancing interface and a wide range of production quality quantum-chemical procedures, allowing for benchmarking different algorithms in a realistic and consistent environment.

The paper is structured as follows. The core computational quantum chemistry algorithms are introduced in Section 2. In the next section the parallel Hartree-Fock calculations are analyzed. In Section 4 the parallel Moller-Pleset Perturbation Theory implementation is discussed. The paper is concluded with short summary in Section 5.

2 CORE ALGORITHMS OF COMPUTATIONAL QUANTUM CHEMISTRY

Computational quantum chemistry can be considered to be just a glorified name for simple tensor algebra. Specifically, these are various transformations of the two-electron integrals (electron repulsion integrals, ERI) tensor which consitute the rate-determining step of the calculations. The ERI are defined as

$$(\mu\nu|\kappa\lambda) = \iint \mathrm{d}\boldsymbol{r}_1 \mathrm{d}\boldsymbol{r}_2 \chi_\mu(\boldsymbol{r}_1)\chi_\nu(\boldsymbol{r}_1)\frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_2|}\chi_\kappa(\boldsymbol{r}_2)\chi_\lambda(\boldsymbol{r}_2) \tag{1}$$

where χ denote the atomic orbitals. It has to be noted that the sheer size of the ERI tensor and its sparsity pattern prevent standard algebraic treatment.

While the actual resource requirements depend strongly on the type of tensor contraction performed by a specific algorithm, we can distinguish two main types of them, the two-index contraction

$$G_{\mu\nu} = \sum_{\kappa\lambda} P_{\kappa\lambda} \left[2(\mu\nu|\kappa\lambda) - (\mu\lambda|\kappa\nu)\right] \tag{2}$$

and the four-index transformation

$$(ia|jb) = \sum_{\mu\nu\lambda\sigma} C_{\mu i} C_{\nu a} C_{\kappa j} C_{\lambda b} (\mu\nu|\kappa\lambda). \tag{3}$$

The former consitutes the core of the Hartree-Fock [3, 4] and hybrid Kohn-Sham [5] methods. The latter is crucial in the post-Hartree-Fock methods for which we used the Moller-Pleset second order perturbation theory (MP2) [4] as a representative example.

In the next sections we analyze how the structure of the key transformations affects the performance of the various load-balancing schemes and propose effictient solutions.

3 PARALLEL HARTREE-FOCK CALCULATIONS

The bottleneck of the two-index contraction of Equation (2) is the two-electron integral generation step, being responsible for roughly 80 % of the whole calculation time. This is caused by both the sheer number of the integrals and by the relatively high computational cost of calculating an integral. The latter holds even for modern quasi-optimal algorithms of integral generation. To considerable extent, the same applies to the Kohn-Sham (KS) calculations with hybrid exchange-correlation functionals.

The structure of the problem results in the natural partitioning scheme which splits the ERI generation into batches and scatters them across the nodes.

3.1 CPU-Time Driven Load-Balancing

We start the analysis with the simplest possible load-balancing algorithm, the static one. It splits the computational problem into equally sized tasks which are uniformly distributed between nodes. While conceptually easy, the approach is bound to be very inefficient. Even assuming highly uniform computational enviroment, the time spent by the processors on performing assigned tasks will be significantly different. This is because the time necessary to calculate integrals involving orbitals with various angular momentum differs significantly.

Recently we proposed an efficient load-balancing algorithm, which dynamically adapts to the inhomogeneity of either the computing environment or the data being calculated [6]. The proposed algorithm works as follows. The computational problem is divided into tasks of different sizes. The tasks are stored in a task queue. A node gets the next task from the queue as soon as it completes the previous one. The splitting of the problem into tasks is organized as follows. A fraction $1/f$ of the original problem is divided into n tasks, where f is the splitting factor and n stands for the number of nodes. Then the procedure is repeated recursively for the remaining part of the problem. The recurrence is stopped when the size of the remaining part is smaller than the threshold t. The algorithm is parametrized by f and t.

Test calculations were performed for sexithiophene (see Figure 1) at the Hartree-Fock level of theory in the 6-31G** basis. The size of the model system is best characterized by the number of orbitals. In this case the number of orbitals is 514, which is of the magnitude typical for commonly performed quantum-chemical calculations. The scaling of the proposed load-balancing algorithms with the number of CPUs is shown in Figure 2. The dynamic load-balancing algorithms performs consistently better than the static one.

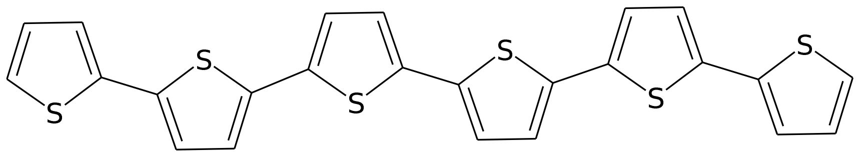


Fig. 1. The sexithiophene molecule

The relative time required to perform calculations using n CPUs may be described by

$$T_n = \frac{\alpha}{n^\beta} + \gamma \tag{4}$$

where α represents the fraction of the parallelized part and γ stands for the (effectively) serial fraction of the calculations. β describes the deviation from linear speedup. The scaling function described above was fitted to the datapoints obtained for the model system. The fitted parameter values are presented in Table 1.

Algorithm	α	β	γ
Static	0.92	0.69	0.07
Dynamic	0.96	1.00	0.03

Table 1. Fitted parameter values for the load-balancing algorithms. See text for details.

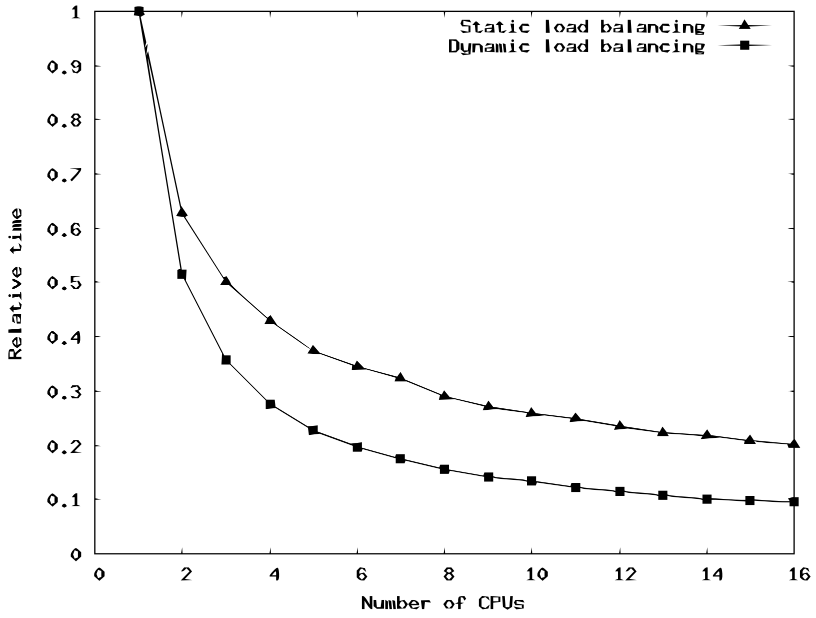


Fig. 2. Time of the SCF calculations for the model system as a function of the number of CPUs. The time is relative to the 1 CPU case. See text for details.

The non-zero value of the γ parameter stems from both non-parallelized parts of the code (mainly the Fock matrix diagonalization) and, to a lesser extent, from the communication overhead. The difference between the values for the static and dynamic dispatcher seems to be mainly a consequence of the relatively poor fit of the γ parameter. Still, in both cases the value is small, which shows that the bulk of the code is executed in the parallel mode.

The main difference between the algorithms is the value of the β parameter, which decides how much faster works the parallelized part of the code with increasing number of the computational nodes. While the scaling for the static case seems to be poor, for the dynamic case it has reached the theoretical limit of 1. However, the performance of the dynamic load-balancing algorithm can still be improved by taking advantage of the hyper-cache effect.

3.2 Integral Cache

To further improve the efficiency of the integral contraction, some of the quantum-chemical programs use integral caching. The cache mechanism relies on storing already computed integrals in fast memory. As the same integrals are used in each iteration of SCF cycle there are significant profits to be realized if their values are cached in the fast memory efficiently. However, for real systems the total number of integrals is too large to allow for keeping them all in the fast memory. Therefore, only a fraction of the integrals can be stored in the cache. The amount of cached integrals depends on the accessible memory.

The caching strategy used in this paper is a very simple one. The integrals are stored in the order they were requested by the Fock matrix generation procedure until the assigned memory is exhausted. Other, prospectively more effective approaches are possible, like giving preference to the integrals involving orbitals of high angular momentum.

Analogous technique can be applied to other computationally intensive procedures, like three-electron integrals used for the density-fitting and atomic orbital values used for exchange-correlation potential generation. However, in this paper we concentrate on the ERI cache.

3.3 Cache Affinity

In the case of single processor, caching is conceptually simple but limited due to usually small memory to use for the purpose. n-node calculations offer n times extended cache memory space and potentially considerable time savings are possible if the load-balancing algorithm allows to use it in an efficient way. The efficiency is determined by cache affinity, which decides how large is the probability that if an integral was processed by given node, it will be processed by the same node in the next SCF cycle.

From the point of view of the cache affinity, the static load-balancing algorithm is optimal. This is because the integrals in the consecutive SCF cycles are always processed by the same nodes as in the previous cycles. On the other hand, the dynamic load-balancing algorithm represents the case of minimal cache affinity, with the probability of an integral being processed by the same node being equal to $1/n$. However, despite the low cache affinity, the dynamic algorithm performs much better than the static one. Therefore, we introduce cache-awareness to the dynamic algorithm to improve its cache affinity without loosing its adaptivity.

The cache-aware load-balancing algorithm works as follows. The computational problem is divided into tasks of different sizes. The tasks are bundled into groups of size n where n stands for the number of nodes. The task groups are stored in the task groups queue. A node gets the next task from the current task group as soon as it completes the previous one. The task to be handed out to the node i is taken from the i-th slot if available or chosen randomly otherwise. When no more tasks are available in the current group, the group is retired and the next one is taken from the queue.

The splitting of the problem into tasks is organized as follows. A fraction $1/f$ of the original problem is divided into n tasks, where f is the splitting factor. The tasks are grouped together, and the group is put to the queue. Then the procedure is repeated recursively for the remaining part of the problem. The recurrence is stopped when the size of the remaining part is smaller than the threshold t. The algorithm is parametrized by the values of f and t.

The algorithm was implemented in Niedoida. Test calculations were performed for the same model system as in Section 3.1.

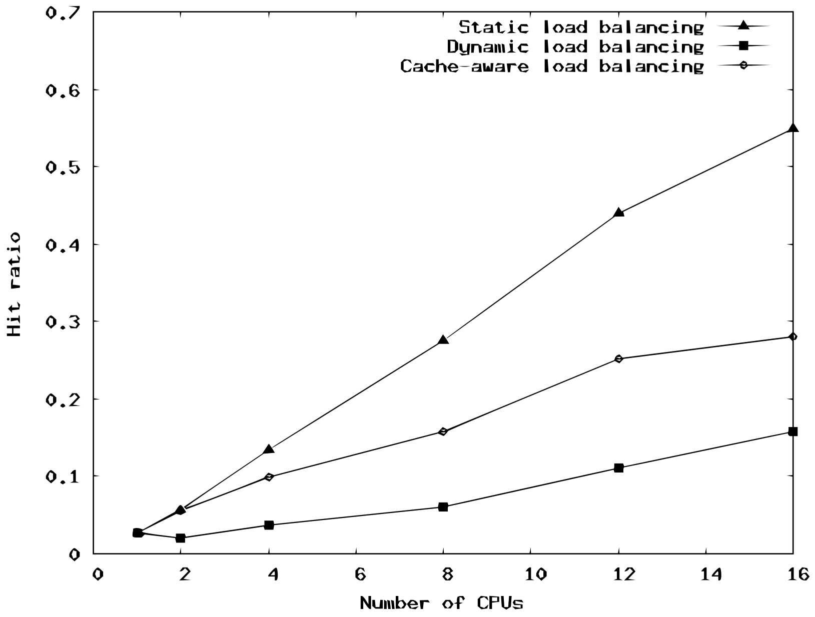


Fig. 3. Cache hit ratio as a function of the number of CPUs. Sexithiophene molecule, HF/6-31G** level of theory, 256 MB of cache memory per CPU.

The cache affinity of a load-balancing algorithm can be characterized by the cache hit ratio. Cache hit ratio is the fraction of the integrals which were retrieved from the cache to the total number of integrals required to perform the whole SCF process. The dependence of the cache hit ratio on the number of computational nodes for the cache-aware algorithm, while worse than for the static one, is significantly better than for the dynamic scheme (see Figure 3).

Comparison of the speed of the analyzed algorithms shows that for small cache sizes the cache-aware algorithms performs on par with the dynamic one. With the growing cache size, the dynamic algorithm performs slightly worse and the cache-aware algorithm performs better. For large cache size (larger than 256 MB per node) the cache-aware algorithm is the fastest one.

To analyze the scaling of the cache-aware algorithm in more detail we performed calculations for the model system using large cache size (512 MB per node). The relative time of the calculations is depicted in Figure 4. Fitting the parameters of Equation (4) yields $\alpha = 1.00$, $\beta = 1.07$ and $\gamma = 0.02$. Comparison with Table 1 shows that the cache-aware algorithm scales better than the other analyzed algorithms. The value of β exceeding the theoretical limit of 1 suggests presence of the hyper-cache effect.

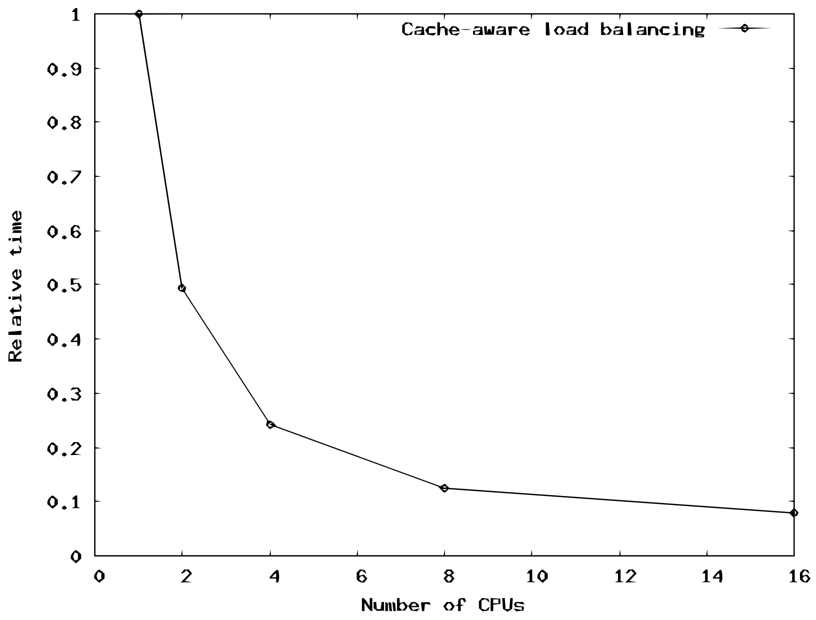


Fig. 4. Time of the SCF calculations for the model system as a function of the number of CPUs. The time is relative to the 1 CPU case. See text for details.

4 PARALLEL MP2 CALCULATIONS

Naively, the time complexity of the transformation of Equation (3) reaches $\mathcal{O}(N^8)$. It can be reduced to $\mathcal{O}(N^5)$ using the four-stage algorithm

$$\begin{aligned}
&\text{for all } i,a,j,b \\
&(i\nu|\kappa\lambda) = \sum_{\mu} C_{\mu i}(\mu\nu|\kappa\lambda) \\
&(ia|\kappa\lambda) = \sum_{\nu} C_{\nu a}(i\nu|\kappa\lambda) \\
&(ia|j\lambda) = \sum_{\kappa} C_{\kappa j}(ia|\kappa\lambda) \\
&(ia|jb) = \sum_{\lambda} C_{\lambda b}(ia|j\lambda).
\end{aligned}$$

However, the reduction of the time complexity comes at the cost of memory complexity reaching $\mathcal{O}(N^4)$ as the tensors of partially-transformed integrals have to be kept in memory.

Given the memory available in currently used computer architectures and the size of typical calculations, the four-stage transformation has to be split into several passes P

```
for all $P$
        for $i \in P$
                for all $a, j, b$
                        $(i\nu|\kappa\lambda) = \sum_{\mu} C_{\mu i}(\mu\nu|\kappa\lambda)$
                        $(ia|\kappa\lambda) = \sum_{\nu} C_{\nu a}(i\nu|\kappa\lambda)$
                        $(ia|j\lambda) = \sum_{\kappa} C_{\kappa j}(ia|\kappa\lambda)$
                        $(ia|jb) = \sum_{\lambda} C_{\lambda b}(ia|j\lambda).$
```

This way the memory consumption is reduced to $\mathcal{O}(N^3)$. Such division of the whole transformation into passes maps directly to the natural partitioning of the problem into parallel tasks.

The characteristic feature of the partitioning is that large part of the two-electron integrals tensor is calculated in every pass. Therefore, contrary to the Hartree-Fock case, CPU-time for integral generation averages out effectively, leaving the time of the single pass to depend solely on the number of orbitals treated. Hence, assuming homogeneous execution environment, the static load-balancing algorithm should be optimal.

Test calculations were performed for linear chain of 15 water molecules in the STO-3G basis. The scaling of the proposed load-balancing algorithms with the number of CPUs is shown in Figure 5.

Fitting the parameters of Equation (4) yields $\alpha = 0.94$, $\beta = 1.06$ and $\gamma = 0.05$. The modest super-linear scaling is attributed to the hyper-cache effect.

5 SUMMARY

Analysis of the key aspects affecting parallel performance of the core computational quantum chemistry algorithms was performed. It has been shown that despite the superficial similarity between the considered algorithms, their parallel form requires different load balancing schemes to yield an efficient solution. The analysis allowed us to design close to optimal load balancing procedures for the studied families of quantum-chemical calculations.

It has been shown that explicit introduction of the memory affinity to the load balancing procedures allows for superlinear scaling. We were able to exploit the hypercache effect in all computational quantum-chemistry algorithms considered in this work.

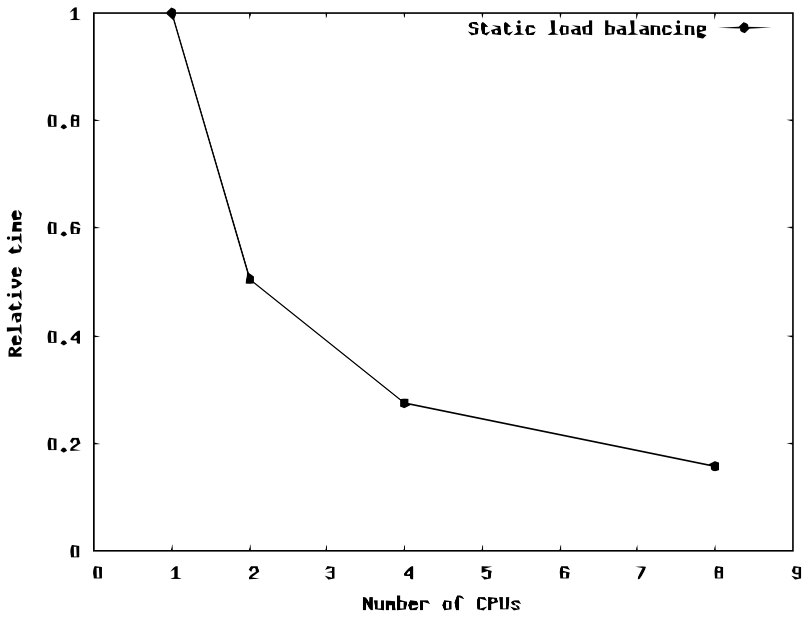


Fig. 5. Time of the MP2 calculations for the model system as a function of the number of CPUs. The time is relative to the 1 CPU case. See text for details.

Acknowledgments

Support from ACK Cyfronet (grant MEiN/SGI3700/UJ/077/2006) is gratefully acknowledged.

REFERENCES

[1] Mazur, G.—Makowski, M.—Piskorz, W.—Ćwiklik, Ł.—Sterzel, M.—Radoń, M.—Kulig, W.—Jagoda-Ćwiklik, B.—Błażewicz, D.: Niedoida 0.3. 2007.

[2] MPI-2: Extensions to the message-passing interface. `http://www-unix.mcs.anl.gov/mpi/mpi-standard/mpi-report-2.0/mpi2-report.htm`.

[3] Roothaan, C. C. J.: New Developments in Molecular Orbital Theory. Rev. Mod. Phys., Vol. 23, 1951, p. 69.

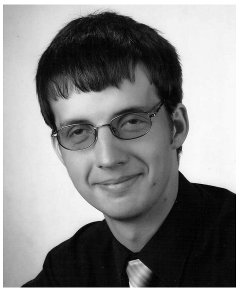
[4] Helgaker, T.—Jorgensen P.—Olsen, J.: Molecular Electronic Structure Theory. John Wiley and Sons, Ltd., 2000.

[5] Becke, A. D.: Density-Functional Exchange-Energy Approximation With Correct Asymptotic Behaviour. Phys. Rev. A, Vol. 38, 1988, p. 3098.

[6] Mazur, G.—Makowski, M.: Development and Optimization of Computational Chemistry Algorithms. Computing and Informatics, Vol. 28, 2009, p. 115.

Grzegorz Mazur is an Assistant Professor at the Department of Computational Methods in Chemistry of Jagiellonian University in Cracow (Poland). He attained his Ph. D. in chemistry in 2001 at the same university. His current research interests include theoretical description of the excited states properties and optimization of quantum chemistry algorithms.

Marcin Makowski is an Assistant Professor at the Department of Theoretical Chemistry of Jagiellonian University in Cracow (Poland). He attained his M. Sc. degree in 2001 and his Ph. D. in 2004, both in chemistry, at the same university, and his B. Sc. degree in computer science at University of Mining and Metallurgy in Cracow in 2004. His current research interests include theoretical molecular spectroscopy, linear scaling methods in quantum chemistry and optimization of quantum chemistry algorithms. He participates in several research projects related with the development of theoretical chemistry formalisms and numerical methodologies that allow to efficiently calculate electronic structure of large molecular systems.



Mateusz Brela is working towards his M. Sc. Degree in theoretical molecular spectroscopy from the Faculty of chemistry of Jagiellonian University in Cracow (Poland). His current research interests include optimization of quantum chemistry algorithms, linear scaling methods in quantum chemistry, theoretical molecular spectroscopy and molecular dynamics.