Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Enabling Large-Molecule Simulations of Biological Interest through LSDALTON's DFT Method

Soon-Heum Ko[a]*, Simen Reine[b], Thomas Kjærgaard[c]

[a]*National Supercomputing Centre, Linköping University, 581 83 Linköping, Sweden*
[b]*Centre for Theoretical and Computational Chemistry, Department of Chemistry, Oslo University, Postbox 1033, Blindern, 0315, Oslo , Norway*
[c]*qLEAP – Center for Theoretical Chemistry, Department of Chemistry , Aarhus University , Langelandsgade 140, Aarhus C , 8000 , Denmark*

Abstract

In this paper, we present the performance of LSDALTON's DFT method in large molecular simulations of biological interest. We primarily focus on evaluating the performance gain by applying the density fitting (DF) scheme and the auxiliary density matrix method (ADMM). The enabling effort is put towards finding the right build environment (composition of the compiler, an MPI and extra libraries) which generates a full 64-bit integer-based binary. Using three biological molecules varying in size, we verify that the DF and the ADMM schemes provide much gain in the performance of the DFT code, at the cost of large memory consumption to store extra matrices and a little change on scalability characteristics with the ADMM calculation. In the insulin simulation, the parallel region of the code accelerates by 30 percent with the DF calculation and 56 percent in the case of the DF-ADMM calculations.

1. Introduction

The advent of Tier-0 systems in Europe nowadays facilitates the ab-initio based evaluation of large biological molecules. The DALTON family of codes is renowned as a both accurate and versatile tool for electronic-structure calculations, which made it to be widely used in life-science community such as the ScalaLife project [1] under an FP7 program. The DALTON code [2] has been developed since 1983 and more than 60 developers contribute to the improvement and new functionalities. The code is distributed freely, under the personal/site license agreement.

Two major branches exist under the DALTON code: traditional DALTON and LSDALTON (Linear Scaling DALTON). The latter was born with the ultimate research goal of deploying a linear-scaling Density Functional Theory (DFT) [3] in order to treat large molecular systems. LSDALTON therefore serves as a framework to test and develop new linear-scaling methods. The novelty of the LSDALTON program made it fairly easy to fully exploit the latest Fortran programming schema and tune it for modern computing architectures. Due to the different application regimes of DALTON and LSDALTON, some fundamental design decisions are changed. LSDALTON does not use point group symmetry, since most large molecular systems have no such symmetry. LSDALTON does however exploit integral-screening techniques that become increasingly important with molecular size, in combination with integral-acceleration techniques and highly efficient integral approximations. LSDALTON is fully atomic-orbital (AO) based, allowing for asymptotic linear-scaling treatment of large molecular systems. DALTON, on the other hand, is molecular-orbital (MO) based, and linear scaling is thus prohibited without major revisions of the code. Even though there are several differences between the two codes, many of the code components are mostly identical and could be shared, like the numerical schemes for the exchange-correlation functional evaluation as needed for the DFT.

* Corresponding author. *E-mail address*: sko@nsc.liu.se

AO-based treatment [4,5] allows for linear-scaling wave-function optimization, molecular gradients and various response-function calculations. As a result, LSDALTON provides better efficiency over the traditional DALTON code on large molecular systems suitable for large-scale parallel simulations.

In this work, we aim at examining the performance of the linear-scaling DFT method in the LSDALTON code in order to enable the simulation of large biological molecules. Both OpenMP and MPI parallelism were introduced to LSDALTON through the prior PRACE work [6], and the scalability was benchmarked on selected small problem sets. In this paper, we make an intensive investigation on the code performance at various MPI sizes and for different molecules. We primarily focus on finding obstacles associated with the density-fitting (DF) and the auxiliary-density-matrix method (ADMM), because these two approximations are essential for achieving highly efficient computations of large biological molecules in Tier-0 scale.

The present paper is structured as follows. The DFT method and associated schemes are introduced in Section 2. The performance of the LSDALTON code in biological molecular simulations is presented in Section 3. In this section, we address the enabling work to overcome the memory-related errors and provide the parallel performance of the DFT code with regular, DF, and ADMM schemes. A short summary and the future work follow in the last section.

2. DFT Method in LSDALTON Code

The DFT is the most popular computational-chemistry method. The success of the DFT stems from the fact that it gives the best compromise between accuracy and computational effort. However, applying the DFT to large molecular systems is challenging and requires efficient parallelization in order to accomplish this goal. A detailed description of the DFT and its implementation is beyond the scope of the paper. However, focusing on the computational aspects, a standard DFT calculation can be characterized as an iterative scheme. Each step in this iterative procedure requires the construction of an AO-based Kohn-Sham matrix and the construction of a new AO density matrix (or a corresponding set of MOs). The construction of a Kohn-Sham matrix requires the calculation of AO integrals while the construction of a new density matrix can be cast in terms of linear algebra routines.

The DF is a method to approximate the electron-repulsion integrals. In effect, the expensive four-center two-electron integrals are replaced by the evaluation of two- and three-center two-electron integrals and the solution of a set of linear equations. The resulting speed-up can be substantial, typically one to two orders of magnitude. However, the price for the reduced cost is an increased memory usage, since the so-called metric matrix (the two-center two-electron integral matrix in the larger auxiliary basis set which is used to approximate the electron density) is typically 9-15 times larger than the conventional Kohn-Sham and density matrices.

The ADMM [7] is a method to approximate the expensive exact Hartree-Fock exchange contribution to the hybrid DFT calculations. The basic idea of the ADMM is to calculate the exact Hartree-Fock exchange contribution in a reduced basis set, using an auxiliary density matrix that is smaller in size. The assumption behind the approximation is that the difference in the exchange energy between the primary and auxiliary density matrices is well captured by a GGA functional. The ADMM scheme should therefore have roughly the same memory consumption as the regular calculation. The ADMM and the DF schemes can be combined so that the DF is used to approximate the Coulomb contribution to the Kohn-Sham matrix and the ADMM is used to approximate the exact exchange contribution to the Kohn-Sham matrix.

The parallelization of the Basic Linear Algebra Subprograms (BLAS) and the Linear Algebra PACKage (LAPACK) routines was done by linking to the parallel-distributed libraries Parallel BLAS (PBLAS) and Scalable LAPACK (ScaLAPACK) [8]. The parallelization of the integrals was done using method-specific MPI/OpenMP schemes [6] and good scalability was demonstrated for up to 2048 cores using pure DFT calculations. However, looking at the strong scaling, the efficiency is determined by the LSTENSOR Fortran derived type. The LSTENSOR structure was used to store all data inside the integral driver. A general integral code uses a variety of input and output arrays, like the two-dimensional AO density matrix, the four-dimensional two-electron integrals, the five-dimensional differentiated two-electron integrals, the two- and three-dimensional DF integrals, etc. LSTENSOR was therefore constructed to be a common structure that could contain all the data needed for input and output handling. However, a large percentage of the time was spent in setting up the LSTENSOR structure, transforming data from the memory-distributed format used by ScaLAPACK to the LSTENSOR structure and back, and accessing the data in the LSTENSOR structure within the integral code. The LSTENSOR limitations on scalability was found to be exaggerated when using the DF techniques and the ADMM, and the initial benchmark results of this project therefore revealed the need to abandon the LSTENSOR structure in favor of simple Fortran array structures.

3. Biomolecular Simulations with LSDALTON

A. Computing Resources

The CURIE [9] supercomputer is one of PRACE (Partnership for Advanced Computing in Europe [10]) Tier-0 resources. It is an x86-64 cluster system that holds in total 80640 CPU cores on the thin node partition. Each node contains two eight-core Intel Sandy Bridge processors and has 64 GB RAM capacity. The system operates on bullx Linux Server 6.1 whose kernel version is 2.6.32. Relatively long-running and large memory-requested Insulin simulations were run on this system.

Another x86-64 supercomputer, named Triolith [11], was also used for performance evaluation of valinomycin and titin simulations. This supercomputer serves as one of the national SNIC (Swedish National Infrastructure for Computing [12]) systems. This system is quite similar to CURIE in terms of both hardware specifications and software stack. The total of 1600 HP SL230s compute nodes are in service, each of which is equipped with two eight-core Intel E5-2660 (2.2 GHz Sandy Bridge) processors. The operating system is CentOS 6.5 whose kernel version is 2.6.32. A noticeable difference from CURIE lies in the RAM capacity per node. Each node of Triolith contains 32 GB RAM, which is half the size of each CURIE standard node. This prevented us from running the Insulin simulation which uses ~60 GB RAM on the DF calculations.

B. Verification of the Density Fitting Scheme on the Valinomycin Molecule

LSDALTON was used to obtain the DFT density and energy for the valinomycin molecule. Valinomycin is an antibiotic that consists of 168 atoms; composed of H (hydrogen), C (carbon), O (oxygen) and N (nitrogen). We used the cc-pVDZ[b] basis set, giving a total amount of 1542 basis functions. For the DF calculation, we used the df-def2 fitting basis set, giving a total of 7518 auxiliary basis functions. For the ADMM calculation, we applied the 6-31G basis, which gives 882 basis functions for the smaller basis. The hybrid functional B3LYP [13,14] was used throughout, which combines Becke's formulation for the exchange part [15], Lee, Yang and Parr's formulation for the correlation part [16] and 20% HF-type exchange.

Four different sets of simulations were performed to study the performance of LSDALTON's DFT simulation: pure MPI and hybrid MPI+OpenMP calculations each using ScaLAPACK/PBLAS or LAPACK/BLAS mathematical libraries. The memory requirement per an MPI rank was a little larger than 2 GB, which exceeded the memory-per-core of the Triolith compute node. Thus, only half of the CPU cores were used so as to secure the extra amount of memory in the case of a pure MPI run, whereas the entire CPU cores in a node could be used in the case of a hybrid simulation. The memory consumption for storing the metric matrix for the DF scheme was close to 0.5 GB.

Total execution times for pure MPI and hybrid simulations are presented in Fig. 1 and Fig. 2. Scalability of the DFT code is experimented over three different numerical configurations: regular (non-DF), DF, and DF combined with ADMM calculations. In these figures, the gray solid line denotes the ideal scalability. Both x- and y-axes are presented in logarithmic scales. All experiments are run from 1 to 32 Triolith compute nodes.

Figure 1 presents the performance of the pure MPI simulation. The DFT simulation iterates until the convergence is reached. In the valinomycin simulation, the Self-Consistent Field (SCF) energy-optimization procedure converges after 17 iterations, regardless of the level of approximation. The DF-ADMM DFT calculation takes the shortest execution time and the DF DFT calculation is the next. This is since the DF-ADMM calculation simplifies the Hartree-Fock exchange calculation on top of the DF formulation. The DF DFT and the regular DFT runs scale similarly and they are highly scalable. The DFT run with the DF-ADMM scheme somewhat scales worse than the other two simulations. When using the ScaLAPACK/PBLAS library for matrix operations, the scalability is much better than using the LAPACK/BLAS library.

Figure 2 presents the performance of the hybrid simulation. The graph shows the similar pattern as the result of the pure MPI simulation. The DF-ADMM DFT calculation converges fastest, followed by the DF DFT calculation. In similar to the MPI simulations, the DF DFT and the regular DFT runs show the similar scalability and they scale better than the DF-ADMM DFT calculation. Additionally, the use of the ScaLAPACK/PBLAS libraries contributes to improving scalability, especially as the number of cores increases. The scalability suddenly becomes far worse at 512 (64 MPI ranks × 8 threads) cores, presumably because of the small problem size. In most cases, the hybrid simulation executes faster than the pure MPI task that runs at the same number of

[b] correlation-consistent polarized valence-only basis sets with double-zeta, first proposed by Dunning

nodes, because half of the CPU cores are sacrificed in pure MPI simulations due to the large memory consumption. The pure MPI simulation runs faster than the hybrid counterpart in the case of the DF-ADMM DFT simulation with the ScaLAPACK/PBLAS on 32 nodes.

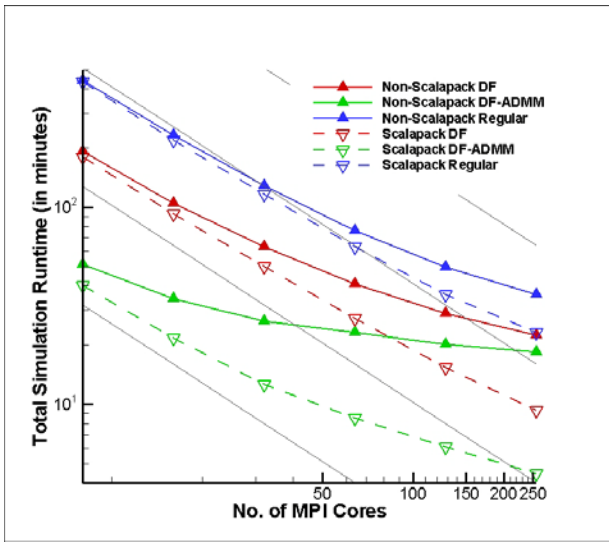

Figure 1 Execution Time for the MPI Simulation of a Valinomycin Molecule. Gray solid lines denote the ideal condition.

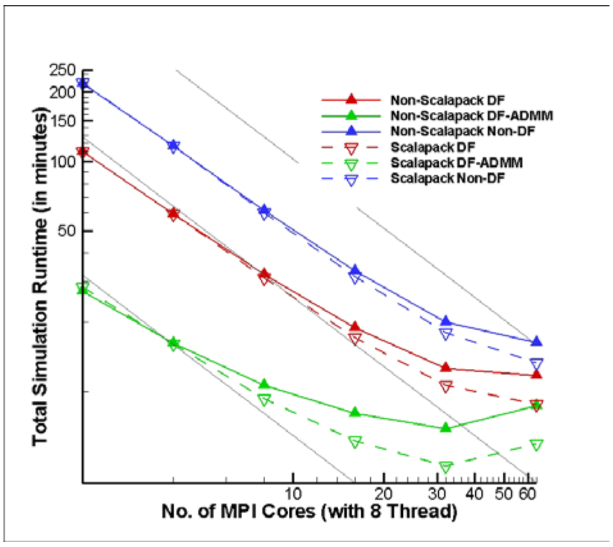

Figure 2 Execution Time for the Hybrid (MPI+OpenMP) Simulation of a Valinomycin.

From the valinomycin simulation, we can draw the following conclusions.

- The DF-ADMM computation saves a significant computation time. On the other hand, it has the poorer scalability over other two methodologies.
- The use of the ScaLAPACK/PBLAS library is essential in terms of scalability and performance.
- The hybrid simulation is preferred to the pure MPI simulation, since a number of CPU cores are sacrificed in MPI simulations due to a large memory requirement in DF simulations.

C. Enabling Large Biomolecular Simulations within LSDALTON

Enabling a TRUE 64-bit integer binary is not at all straightforward. Utilized libraries do have their own limitation and finding a working composition is a laborious task.

With regard to the MPI library, OpenMPI was one of the software that supports the communication of large datasets over 2 GB.[c] System-supplied commercial libraries were first experimented because they encapsulate several configurations in the single package and users can easily link the right library by simply adding proper compilation options. However, in the case of Intel MPI, it fails to perform the message passing arrays larger than 2 GB since the message size parameter is fixed as 32-bit integer. Bullxmpi installations on the CURIE supercomputer were defining Fortran integers as 32-bit, thus naturally leading to failure in handling large size arrays. Open-source OpenMPI library fully supports message passing of large datasets if it is installed with 64-bit integer representation, though more care is needed in compiling user's code (i.e., matching the compiler with its version, datatype declaration). More detail is described in Appendix A: Issues with MPI Library.

The linear algebra library that takes care of matrix operation also raised the confusion. Both LSDALTON's built-in matrix operator and Intel MKL's LAPACK/BLAS interface were stable regardless of the matrix size. On the other hand, MKL's ScaLAPACK/PBLAS library with 64-bit interface did not work as desired: it worked successfully in the valinomycin simulation which handles the array within 32-bit integer range, while a memory error takes place in titin/insulin simulations which create large matrices over 2GB in size. Detailed description is provided in Appendix B: 64-bit Math Libraries.

Therefore, the compilation environment that performed best is as follows:

- Intel compiler, version 14.0.2 (Intel Composer XE 2013 SP1 Update2)
- OpenMPI, version 1.6.5
- Intel MKL's LAPACK/BLAS 64-Bit Interface, version 11.1.2 (Intel Composer XE 2013 SP1 Update2)

D. Titin and Insulin Simulations using DFT Method

The DFT performance of LSDALTON was thoroughly investigated for large biological molecules of titin and insulin. A titin consists of 392 H, C, O, N, or S (sulphur) atoms. Used regular basis set was the 6-31G* type[d] and a total of regular basis functions is 3196. The auxiliary basis set was the df-def2 type and has 18761 functions. The 3-21G basis set with 2196 basis functions was also utilized for the ADMM computation. An insulin is composed of 787 H, C, O, N, or S atoms. The cc-pVDZ regular basis set with 7604 functions and the df-def2 auxiliary basis set consisting of 37853 basis functions were used. The STO-3G and the 6-31G bases are used as complimentary auxiliary basis function and JK auxiliary basis function, respectively. The STO-3G basis holds 2431 functions and the 6-31G consists of 4433 basis functions. In both titin and insulin simulations, the B3LYP hybrid functional was applied.

Since the requested amount of memory was larger than the memory-per-core of the available computing resources, we only conducted hybrid simulations. The number of threads was set to 8 for the titin simulation, which assigned one MPI rank per each Sandy Bridge processor. In the case of the insulin calculation, the single MPI rank consumed almost all RAM memory of the node. Thus, only one MPI rank was launched per a node in a threaded mode using all 16 cores. We used the MKL LAPACK/BLAS library for matrix operations due to the noted issue in a ScaLAPACK/PBLAS implementation.

Figures 3 to 6 represent the performance for the titin simulation for three different configurations. The code continued running until convergence and the number of SCF iterations were mostly the same among different configurations (DFT calculations with the DF and the DF-ADMM converged after 21 iterations and the regular DFT calculation converged after 20 iterations). The simulations were launched over 4 to 256 MPI ranks, each holding 8 threads.

Computation time per one SCF iteration of DFT calculation is plotted in Fig. 3. A DFT simulation with the DF-ADMM shows the best performance at small MPI sizes and the run with the DF performs best at 1000+ (128

[c] More precisely, the "allocatable" array on the heap memory space fails to communicate if the size exceeds 32-bit representation (231 bit). There exists a controversy that the message passing is successful in case the static array is defined, though no strong verification was performed: http://stackoverflow.com/questions/13211990/mpi-send-error-with-derived-data-type-fortran.
[d] A valence double-zeta polarized basis set that adds to the 6-31G set six d-type Cartesian-Gaussian polarization functions on each of the atoms Li through Ca and ten f-type Cartesian Gaussian polarization functions on each of the atoms Sc through Zn.

ranks × 8 threads and 256 ranks × 8 threads) cores. In terms of scalability, the DF DFT and regular DFT runs show a very similar pattern, while the DF-ADMM DFT calculation scales worse than them. As the number of CPU cores increase to 1000, the performance of all simulations reaches a saturation plateau and the computation time increases at 2048 CPU cores. In DF-ADMM DFT runs, the extra communication cost overtakes the computational gain at 256+ CPU cores. It means that the ADMM implementation in the LSDALTON code is not well tuned for large number of CPU cores. Nevertheless, the DF-ADMM case at 32 × 8 cores takes the shortest time out of all DFT simulations. This proves that the DF-ADMM scheme is very powerful in reducing computational cost.

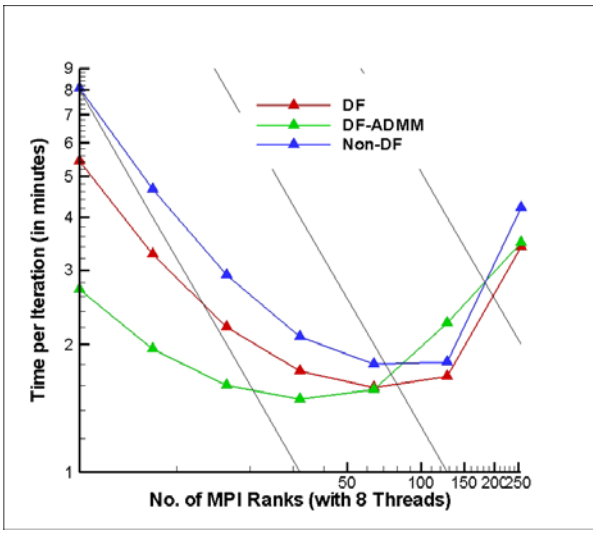

Figure 3 Elapsed Time per One SFT Iteration for the Hybrid Simulation of a Titin Molecule (Time at the 2nd Iteration).

To investigate the parallel performance in detail, we compare the computation time for Kohn-Sham matrix construction, including Coulomb, Exchange and Exchange-Correlation contributions. All other parts of the code in the SCF iteration run serially and are independent of the matrix design. From Fig. 4 we observe that the DF DFT and the regular DFT runs have good scalability up to 32 × 8 CPU cores. The scalability with the DF-ADMM is worse than other schemes and the performance reaches a saturation plateau at 32 × 8 CPU cores. Compared to the regular simulation, the DF DFT computation saves about 30% of computation time in the case of 32 × 8 cores and the percentage increases for smaller MPI sizes. Computation time is the shortest for the DF-ADMM DFT calculation with 32 × 8 cores, where elapsed time is 37.44 seconds. For the DF DFT simulation, the shortest computation time is monitored at 64 × 8 cores with 43.74 seconds. The regular run also shows the best performance at the same number of cores, taking 56.17 seconds.

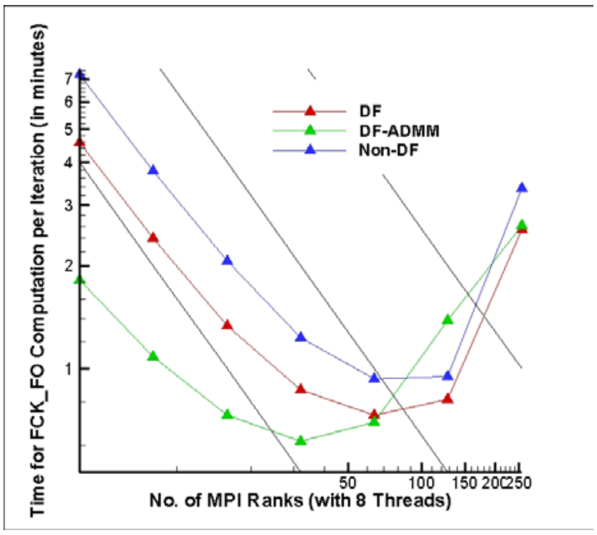

Figure 4 Elapsed Time for the Single Kohn-Sham Matrix Construction for the Hybrid Simulation of a Titin Molecule (Time at the 2nd Iteration).

Figure 5 shows the computational overhead from non-iterative routines. This overhead is measured by subtracting the SCF iteration time from the total DFT execution time. It includes an initial creation of matrices (density matrix components, an auxiliary fit, the ADMM construct, etc) and the global tensor data structure, as well as other general components such as I/O and the initialization. The initial tensor size for the DF simulation becomes much larger than that for the regular computation since it has to store extra components related with the auxiliary basis set. This results in a longer time for creating the tensor structure and broadcasting the entire structure by the master rank. An extra cost for the ADMM simulation is mostly due to the time for producing the exchange-matrix contribution, which is around 1 minute independent of MPI sizes. The overhead tends to increase as the MPI size increases, because many routines at the initial and final stages are serially executed in a master rank and the product is broadcasted to all other MPI ranks. The overhead is larger and grows faster for the DF/DF-ADMM DFT simulations due to broadcasting a large tensor structure.

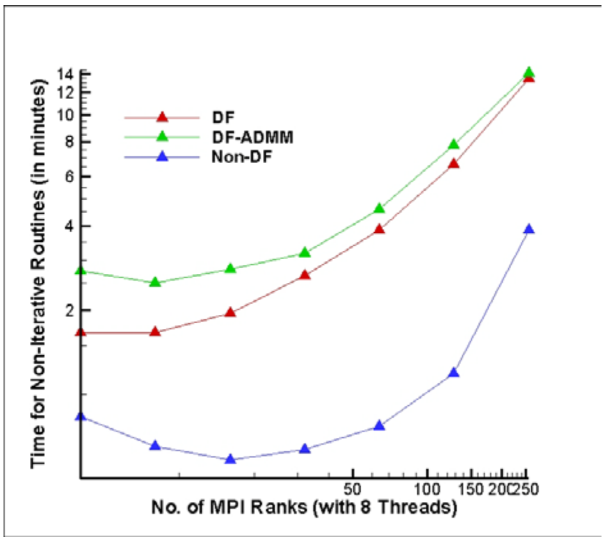

Figure 5 Extra Overhead for the Hybrid Simulation of a Titin.

The total execution time until convergence is presented in Fig. 6. Much of the gain of the DF and the DF-ADMM methods in SCF iterations is largely cancelled out by the initial overhead for the matrix creation and the tensor generation. Still, the DF-ADMM DFT simulation at 32 × 8 cores shows the best performance out of all experiments.

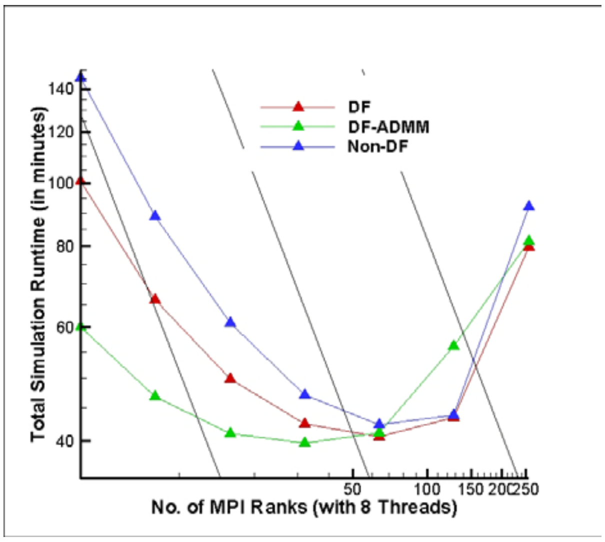

Figure 6 Total Execution Time for the Titin Molecule Simulation.

Figures 7--9 represent the performance for the insulin simulation for three different configurations. Because the insulin simulation takes essentially longer time than that of titin, the code is set to run only 2 iterations. Simulations are launched over 8 to 256 MPI ranks, each holding 16 threads.

Figures 7 and 8 present the DFT computation time per the single SCF iteration and the Kohn-Sham matrix construction. The DF-ADMM DFT calculation shows the best performance over an entire range except 4096 (256 rank × 16 threads) cores. The scalability is quite similar to the case of titin, where the DF DFT and regular DFT runs showed a very similar pattern and the calculation with the DF-ADMM was poorer. Looking at the scalability of Kohn-Sham matrix construction graph, DF DFT and regular DFT runs up to 64 × 16 cores are highly scalable while these runs at 2000+ cores and all DF-ADMM DFT runs are far from scalable. In view of the execution time, the DF-ADMM DFT run at 32 × 16 cores provides the lowest computation time with 3.07 minutes for Kohn-Sham matrix construction. The DF DFT calculation shows the best result of 5.42 minutes at 64 × 16 cores and the regular run was the fastest at 128 × 16 cores by 8.13 minutes.

As the problem size increases, so increases the computation time per iteration. This implies that the amount of saved computation time by the DF or the DF-ADMM approach accordingly increases in comparison to the computation cost in the regular calculation. Moreover, a higher reduction rate of Kohn-Sham matrix construction time is observed in the insulin simulation than the titin simulation. We monitored that the DFT calculation with the DF saved about 30% of the computation time in the case of 32 × 8 cores of the titin calculation: it saves 56% in comparison to a regular DFT simulation (6.90 minutes with the DF scheme and 15.67 minutes with the regular run) at 32 × 16 cores. Indeed, the computation cost is further saved by applying the ADMM approximation. This result expresses that the DF and the DF-ADMM methods are more powerful for the simulation of a large molecular structure in terms of the number of atoms and basis functions, in the condition that the memory requirement does not exceed the hardware capacity.

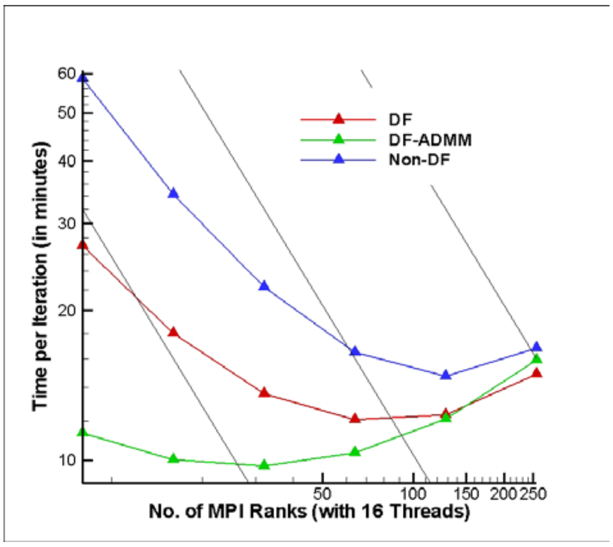

Figure 7 Elapsed Time per One SCF Iteration for the Hybrid Simulation of an Insulin Molecule.

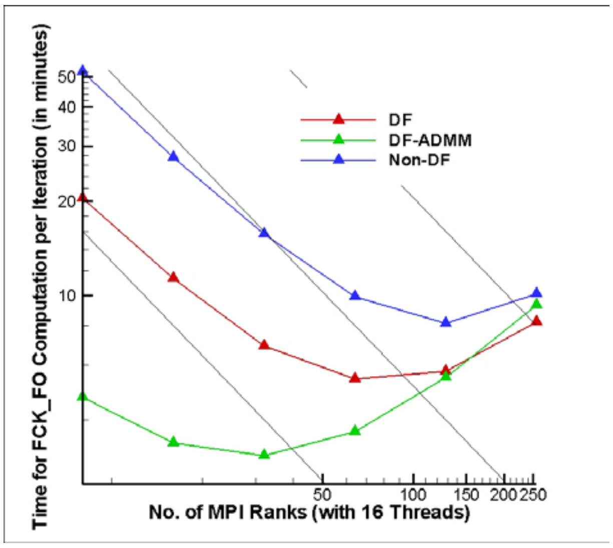

Figure 8 Elapsed Time in Kohn-Sham Matrix Construction for the Hybrid Simulation of an Insulin Molecule (Time at the 2nd Iteration).

The extra overhead in the insulin simulation is reported in Fig. 9. Like the titin simulation, the cost increases as we apply the DF and the ADMM methods. Comparing between the regular DFT and the DF-ADMM DFT cases, the DF-ADMM DFT calculation takes 10 minutes longer at 8 × 16 cores, which grows to be 51 minutes at 256 × 16 cores. In the case of 1024 (64 × 16) cores which looks to be a reasonable MPI size for the insulin simulation, the DF-ADMM DFT calculation saves 6.15 minutes per an SCF iteration (10.35 minutes taken in the DF-ADMM DFT run and 16.50 minutes in the regular DFT run). The extra cost for the initiation is 18.04 minutes (22.32 minutes in the DF-ADMM DFT run and 4.28 minutes in the regular DFT run). It means that the DFT run with the DF-ADMM schemes will be more efficient than the regular DFT calculation, if the total number of iteration until convergence is more than 4. We argue that the extra cost for initialization is acceptable considering the high gain from applying the DF and the DF-ADMM schemes.

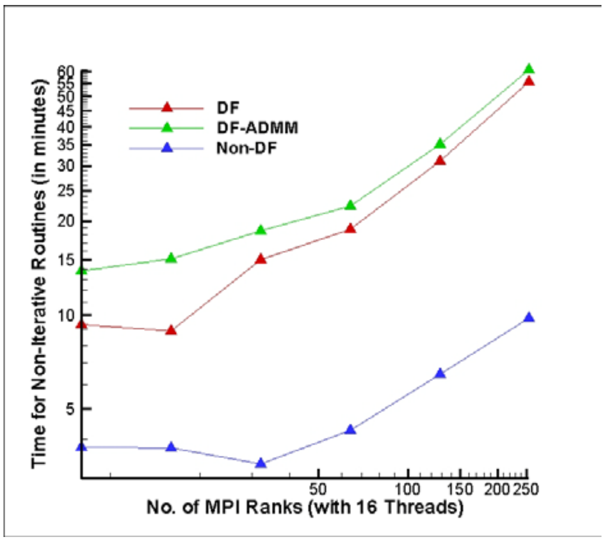

Figure 9 Extra Overhead for Hybrid Simulation of Insulin.

Finally, the amount of allocated RAM memory is presented in Table 1. As seen, the DF computation requires much larger memory allocation than the regular calculation and the size of the allocated array is almost independent of the MPI sizes. Therefore, it is unfortunately very hard to apply the DF scheme to larger biomolecular simulations before the matrix and the tensor data structures are reformulated. The DF-ADMM computation allocates the same memory as the DF method.

Table 1 Memory Consumptions in Hybrid Simulations with 64 MPI Ranks. Number of threads are 8 in valinomycin and titin simulations, and 16 threads used for insulin simulations.

	Density Fitting			Regular (Non-DF)		
	Total	*Matrix*	*Tensor*	Total	*Matrix*	*Tensor*
Valinomycin	2.46 GB	*1.38 GB*	*530 MB*	851 MB	*742 MB*	*148 MB*
Titin	16.04 GB	*7.68 GB*	*3.26 GB*	3.73 GB	*3.19 GB*	*734 MB*
Insulin	62.32 GB	*28.94 GB*	*13.23 GB*	13.69 GB	*13.41 GB*	*2.49 GB*

4. Discussion and Conclusion

In this work, we evaluated the DFT performance of LSDALTON for the simulation of large biological molecules. Primary effort was put towards enabling the DF scheme and the ADMM approximation on top of the DFT method, which contributes much to the performance improvement without harming the convergence of the code.

Initial investigations revealed that the large memory allocation accompanies the usage of the DF scheme of the LSDALTON code, due to the necessity of storing extra matrix components. We tried to solve the memory-related issue by addressing a right composition of the build environment. It was struggling to find MPI and linear algebra libraries who truly support 64-bit integer capability. After investigation of different MPI libraries, we found that OpenMPI was such a library, providing the large message passing over 2 GB. Some LAPACK/BLAS routines are found to capacitate the matrix operation over 2 GB in size, such as Intel MKL and OpenBLAS. Yet, no ScaLAPACK library was found to handle large matrix sizes successfully, without labouring intensive changes to the ScaLAPACK source code. So, OpenMPI and Intel MKL's LAPACK/BLAS interface are selected to be used in conjunction with Intel compiler, to build a truly 64-bit integer-based binary.

We evaluated the performances of the DF scheme and the ADMM approximation by simulating three biological molecular structures of valinomycin, titin and insulin. Notable performance improvements are observed with the DF and the ADMM methods, without noticeable degradation of the convergence criteria. In all three cases, the runtime for the single SCF iteration of the DFT calculation was shorter with the DF scheme and the code accelerated further with the ADMM approximation. In the case of the insulin simulation, the parallel region of the code accelerates by 30 percent with the DF method and 56 percent with the DF-ADMM methods.

With the DF scheme, the scalability pattern remains almost the same as the regular DFT run in small number of cores (less than 1K cores) and the performance with the DF scheme is still better than the regular DFT run despite of a slight loss in scalability at more than 1K cores. We argue that the DF implementation is highly beneficial for most production runs of moderate molecular sizes. With the ADMM approximation on top of the DF scheme, we can normally expect further performance gain at small number of processors. On the other hand, the scalability becomes much poorer with the ADMM scheme: the DFT code does not scale any further at larger than 256 or 512 cores, depending on the size of molecules. The ADMM approach could be recommended if the number of utilized cores remain within this range. At the same time, further investigation is necessary to improve scalability of the ADMM implementation.

The memory requirement of the DF implementation within LSDALTON code is a bottleneck in applying this method to large molecular simulations. The size of allocated memory was around 60 GB per an MPI rank in the case of insulin simulation. Remembering that the DF calculation does not scale any further at more than 2K CPU cores, the use of more than 2K parallel cores is not meaningful on general-purpose cluster systems. Instead, the regular DFT implementation can be applied to the simulation of molecules larger than insulin.

The DF and the ADMM implementations introduce extra overheads in the non-iterative part of the code. It is natural since these schemes allocate extra matrix components for an auxiliary basis set and the resultant cost for broadcasting increases. Yet, the amount of overhead could be reduced by redesigning the internal tensor structure to a lighter fashion. A new tensor design is expected to be released with a future LSDALTON code.

The valinomycin simulation demonstrates that scalability is improved by using a ScaLAPACK/PBLAS library. However, ScaLAPACK/PBLAS libraries up to now fail to allocate large arrays over 32-bit integer range. Since it is not at all trivial to change the ScaLAPACK source code personally, we shall sustain with the LAPACK/BLAS library or lighten the matrix structure of LSDALTON.

Overall, the DF and the ADMM schemes proved to contribute much on increasing the performance of the code. Further changes to reduce the memory consumption and improve the scalability of the ADMM scheme will enable the simulation of larger molecular systems than the current experiments.

Acknowledgements

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

References

[1] http://www.scalalife.eu/

[2] K. Aidas, C. Angeli, K. L. Bak, V. Bakken, R. Bast, L. Boman, O. Christiansen, R. Cimiraglia, S. Coriani, P. Dahle, E. K. Dalskov, U. Ekström, T. Enevoldsen, J. J. Eriksen, P. Ettenhuber, B. Fernández, L. Ferrighi, H. Fliegl, L. Frediani, K. Hald, A. Halkier, C. Hättig, H. Heiberg, T. Helgaker, A. C. Hennum, H. Hettema, E. Hjertenæs, S. Høst, I.-M. Høyvik, M. F. Iozzi, B. Jansik, H. J. Aa. Jensen, D. Jonsson, P. Jørgensen, J. Kauczor, S. Kirpekar, T. Kjærgaard, W. Klopper, S. Knecht, R. Kobayashi, H. Koch, J. Kongsted, A. Krapp, K. Kristensen, A. Ligabue, O. B. Lutnæs, J. I. Melo, K. V. Mikkelsen, R. H. Myhre, C. Neiss, C. B. Nielsen, P. Norman, J. Olsen, J. M. H. Olsen, A. Osted, M. J. Packer, F. Pawlowski, T. B. Pedersen, P. F. Provasi, S. Reine, Z. Rinkevicius, T. A. Ruden, K. Ruud, V. Rybkin, P. Salek, C. C. M. Samson, A. Sánchez de Merás, T. Saue, S. P. A. Sauer, B. Schimmelpfennig, K. Sneskov, A. H. Steindal, K. O. Sylvester-Hvid, P. R. Taylor, A. M. Teale, E. I. Tellgren, D. P. Tew, A. J. Thorvaldsen, L. Thøgersen, O. Vahtras, M. A. Watson, D. J. D. Wilson, M. Ziolkowski, and H. Ågren, "The Dalton quantum chemistry program system," WIREs Comput. Mol. Sci. (doi: 10.1002/wcms.1172)

[3] C. J. Cramer, "Density Functional Theory," in Essential of Computational Chemistry, pp.233—273, John Wiley & Sons Ltd (2004)

[4] P. Salek, S. Høst, L. Thøgersen, P. Jørgensen, P. Manninen, J. Olsen, B. Jansik, S. Reine, F. Pawlowski, E. Tellgren, T. Helgaker, and S. Coriani, "Linear-scaling implementation of molecular electronic self-consistent field theory," J. Chem. Phys. 126, 114110 (2007)

[5] S. Coriani, S. Høst, B. Janisk, L. Thøgersen, J. Olsen, P. Jørgensen, S. Reine, F. Pawlowski, T. Helgaker, and P. Salek, "A linear scaling implementation of molecular response theory in self-consistent field electronic-structure theory," J. Chem. Phys. 126, 154108 (2007)

[6] S. Reine,T. Kjærgaard, T. Helgaker, O. Vahtras, Z. Rinkevicius, B. Frecus, T. W. Keal, A. Sunderland, P. Sherwood, M. Schliephake, X. Aguilar, L. Axner, M. F. Iozzi, O. W. Saastad, and J. Gimenez, "Petascaling and Performance Analysis of DALTON on Different Platforms," PRACE White Paper, Available at http://www.prace-ri.eu/IMG/pdf/Petascaling_and_Performance_Analysis_of_DALTON_on_Different_Platforms.pdf

[7] M. Guidon, J. Hutter, and J. VandeVondele, "Auxiliary Density Matrix Methods for Hartree–Fock Exchange Calculations," J. Chem. Theory Comput., 6 (8), pp.2348-2364 (2010) (doi: 10.1021/ct1002225)

[8] S. Reine, T. Kjærgaard, T. Helgaker, O. W. Saastad, and A. Sunderland, "A ScaLAPACK-based Parallelization of LSDALTON," PRACE White Paper, Available at http://www.prace-project.eu/IMG/pdf/a_scalapack-based_parallelization_of_lsdalton.pdf

[9] TGCC CURIE, http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm

[10] Partnership for Advanced Computing in Europe, http://www.prace-ri.eu/

[11] Triolith User Guide, http://www.nsc.liu.se/systems/triolith/

[12] SNIC Homepage, http://www.snic.vr.se/

[13] K. Kim and K. D. Jordan, "Comparison of Density Functional and MP2 Calculations on the Water Monomer and Dimer," J. Phys. Chem. 98 (40) pp.10089-10094 (1994)

[14] P. J. Stephens, F. J. Devlin, C. F. Chabalowski, and M. J. Frisch, "Ab Initio Calculation of Vibrational Absorption and Circular Dichroism Spectra Using Density Functional Force Fields," J. Phys. Chem., 98 (45), pp.11623-11627 (1994)

[15] A. D. Becke, "Density-functional exchange-energy approximation with correct asymptotic behaviour," Phys. Rev. A, 38 (6), pp.3098-3100 (1988)

[16] C. Lee, W. Yang, and R. G. Parr "Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density," Phys. Rev. B, 37 (2), pp.785-789 (1988)

Appendix. Building a Binary with TRUE 64-bit Integer Support

A. Issues with MPI Library

Compiler technologies are highly advanced that users can easily select the byte representation of datatypes in their serial codes by adding proper compilation flags. On the other hand, compiling MPI-parallelized code sometimes leads to confusion, since MPI library itself is a series of compiled objects under a specific configuration. Situation is more favourable in the case of using commercial MPI libraries (e.g., Intel MPI). They usually encapsulate a series of different configurations under the single package and load the right solution (library and environmental parameters) based on a user's compilation option. On the other hand, open-source MPI libraries (e.g., OpenMPI, MVAPICH) frequently build the single set of libraries suited for the environmental configuration at time of library installation. It results that the code fails to compile or the binary fails to run if the configuration changes after the library was built. Thus, more care is needed to make use of open-source libraries.

Intel MPI (on Triolith at NSC) was first exploited to compile LSDALTON code with 64-bit integer representation. It is simple to compile with 64-bit integer formulation through Intel MPI:

- Replace "*use mpi*" to "*include 'mpif.h'*" from source code. "*use mpi*" loads a pre-built MPI module file which is based on 32-bit integer specification.
- Add "*-ilp64*" flag at time of compilation. Alternatively, this flag can be addressed as the global option at time of launching MPI task, i.e., "*mpirun –ilp64 …*".

We confirmed that the right library (64-bit interface) had been linked. However, the binary still crashed in the middle of execution. The error message was as follows.

```
Assertion failed in file ../../i_rtc_cache.c at line 631: buf_end_palign > buf_start_palign
```

According to online discussion[e], the current limitation of message passing size is 2 GB due to the fact that type 'int' is used by the MPI standard to designate a message size parameter. It expresses that Intel MPI is not a perfect 64-bit integer provider.

Bull-supported MPI on CURIE was also examined to figure out whether it supports the 64-bit integer interface. Since bullxmpi is developed on top of OpenMPI, this library borrows most commands from OpenMPI. The default integer size in Fortran interface is easily detected by running a following command.

```
$ ompi_info -a | grep "Fort integer size"
```

On all bullxmpi installation on CURIE, returned value is 4 byte. Therefore, bullxmpi also fails to create a TRUE 64-bit integer binary for creating and exchanging large arrays.

Personal installation of OpenMPI is the only solution for an end-user. OpenMPI 1.7.5 and 1.6.5[f] were experimented. Intel compiler serves as the baseline compiler for MPI library. The installation command is as follows:

```
./configure --prefix=$USERS_OWN_PATH --with-slurm --enable-mca-no-build=btl-tcp
CC=icc CXX=icpc FC=ifort F77=ifort F90=ifort FFLAGS=-i8 FCFLAGS=-i8

make; make install
```

Since intra-node communication on CURIE is conducted through direct memory access, OpenMPI's default reference to KNEM service (intra-node MPI communication kernel module) shall be disabled. That could be handled by turning off the service in user's environmental configuration[g] or at runtime[h].

[e] https://software.intel.com/en-us/forums/topic/361060

[f] Version 1.6.5 was added for experimentation since scalasca 1.X profiling tool explicitly links to mpi_f77 and mpi_f90 libraries whereas they are deprecated from OpenMPI version 1.7 (Instead these two libraries are merged into mpi_mpifh).

[g] export OMPI_MCA_btl_sm_use_knem=0; export OMPI_MCA_btl_openib_pkey=0x8090;
export OMPI_MCA_btl_openib_ib_service_level=5; export OMPI_MCA_btl_base_exclude=tcp;
export OMPI_MCA_btl_openib_warn_default_gid_prefix=0

[h] mpirun "--mca btl_sm_use_knem 0" …

B. 64-bit Math Libraries

LSDALTON associates lots of matrix operations for the SCF energy optimization. Matrix size grows much bigger at the DF scheme than the typical DFT calculations since the number of auxiliary basis function which is used on the DF calculations is far larger than the regular Gaussian basis function on typical DFT computations. Matrix size easily exceeds 2GB in our experiments on titin and insulin simulations, thus 64-bit integer representation is necessary to solve these molecules.

There exist 3 different ways to conduct matrix operations at LSDALTON code: the call to external ScaLAPACK, the link to sequential/threaded LAPACK and BLAS, and the use of built-in matrix operation functions. As long as they operate as desired, the best choice would be the use of ScaLAPACK since it supports the parallel distributed operation.

Intel MKL math library was linked to exploit ScaLAPACK features. However, the code crashed in a call to PDGEMR2D routine, which provides a copy from any block cyclically distributed (sub)matrix to any other block cyclically distributed (sub)matrix, with the following error message:

xxmr2d: out of memory

From online discussion[i] we find that this error is associated with large matrix array creation which exceeds 2GB in size. Intel argues that this error was resolved at MKL version 10.3[j] but the problem still persists in both MKL 10.3.10 and 11.1.2. We rather decided to make our own installation of open-source ScaLAPACK library[k] after applying recommended changes on REDIST/SRC/pgemraux.c.[l]

Open-source ScaLAPACK requests BLAS and LAPACK libraries as prerequisite. So we first installed OpenBLAS[m] as the BLAS/LAPACK library, which is verified to fully provide 64-bit integer support. However, 64-bit integer ScaLAPACK installation failed to pass its own testsuite, reporting the malfunction in incorporated BLACS (Basic Linear Algebra Communication Subprograms) library. We later linked MKL's OpenMPI BLACS and skipped BLACS installation from ScaLAPACK, but the error persisted. The final decision was made to utilize the sequential/threaded LAPACK and BLAS from Intel's MKL, which successfully worked at any matrix sizes.

In summary, we could confirm that 64-bit array creation is fully supported in MKL's LAPACK and BLAS routines. On the other hand, the 64-bit interface implementation in ScaLAPACK is suspicious. Out of simulations with 64-bit interface, valinomycin was the only successful case which handles less than 1GB matrix. Simulations failed in titin and insulin cases, both of which creates matrix over 2GB.

[i] https://software.intel.com/en-us/forums/topic/509048;
https://software.intel.com/en-us/forums/topic/286499
[j] https://software.intel.com/en-us/articles/intel-mkl-103-bug-fixes,
DPD200199131 PDGEMR2D: out of memory error even when using 64-bit libraries
[k] http://www.netlib.org/scalapack/
[l] https://icl.cs.utk.edu/lapack-forum/viewtopic.php?t=465
[m] http://www.openblas.net/