Scaling Dalton, a molecular electronic structure program

Xavier Aguilar*, Michael Schliephake*, Olav Vahtras*†, Judit Gimenez‡ and Erwin Laure*
*PDC–Center for High Performance Computing and SeRC–Swedish e-Science Research Center
KTH Royal Institute of Technology, Teknikringen 13, SE-100 44, Sweden
†Department of Theoretical Chemistry and Biology
KTH Royal Institute of Technology, Roslagstullsbacken 15, SE - 106 91 Stockholm, Sweden
‡Barcelona Supercomputing Center, Universitat Politecnica de Catalunya,
Barcelona, Spain

***Abstract*—Dalton is a molecular electronic structure program featuring common methods of computational chemistry that are based on pure quantum mechanics (QM) as well as hybrid quantum mechanics/molecular mechanics (QM/MM). It is specialized and has a leading position in calculation of molecular properties with a large world-wide user community (over 2000 licenses issued). In this paper, we present a characterization and performance optimization of Dalton that increases the scalability and parallel efficiency of the application. We also propose a solution that helps to avoid the master/worker design of Dalton to become a performance bottleneck for larger process numbers and increase the parallel efficiency.**

I. Introduction

Computational chemistry deals with calculation of properties and structures of molecules using methods based in mathematical models provided by quantum and classical physics. These computational methods allow scientists to predict and simulate characteristics of chemical systems such as geometrical and electronic structures, energy states, reactivity and spectra. The end applications of computational chemistry play an important role in our current society; for the design of new molecules for new materials, chemicals products or drugs, and for the improvement of existing products or processes to shorten development cycles or to improve energy efficiency.

The methods of computational chemistry are very useful tools but they are associated with high computational costs. The computational challenges that a quantum chemist faces are of various kinds. In this brief overview we shall mention three parts of the computational process which exhibit performance problems for calculations in large systems.

1. The Schrödinger equation is a coupled partial differential equation in $3N$ coordinates, given N particles in the system. The most basic method of quantum chemistry is known as the Hartree-Fock approximation (HF), an independent-particle method where each electron moves in an averaged field of all other electrons, and is the starting point for many other methods. It is generally solved in quantum chemistry, considering only the electrons, by expanding the wave function in sets of one-particle basis functions and using methods of linear algebra. In this way the partial differential equation is transformed into a matrix equation. The total electronic wave function is in this case a single anti-symmetrized product of one-electron functions, and the computational building blocks are one- and two-electron integrals, which have the form of multidimensional arrays of rank two and four, respectively. This means that if the size of the basis set is 10^3 , which is not uncommon, theoretically the number of two-electron integrals is of the order 10^{12}. Rather than storing two-electron integrals on disk, which was the case in early quantum chemistry codes, it is necessary for large systems to recalculate them as needed and throwing them away between iterations, a method known as direct Self-Consistent-Field (SCF) pioneered by Almlöf [1]. The calculation (and re-calculation) of integrals is the time-consuming part of an integral-direct Hartree-Fock calculation.

2. Density Funcional Theory (DFT) is one of the most important methods today [2] being used in more and more applications. It corrects the most important error in the HF methods, namely the lack of electron correlation, by introducing a semi-empirical energy functional. The existence of such a functional has been proven mathematically [3], but its exact form is unknown and there exists a large number of parameterized functionals of different quality. The calculation of the DFT contribution to a property involves a numerical integration over a grid; at every grid point the parameterized wave function is evaluated, combined with a weight factor and added to a total sum.

3. In recent years there has been an increased interest for biological applications among computational chemists. One of the most popular approaches is the so-called QM/MM method that separates a system into two contributions to the properties, a quantum mechanical part and a classical part. [4] The idea is that quantum effects are more short-range than classical interactions, so the influence of a remote part of the system on a subsystem of interest can be approximated by a potential generated by a multi-pole expansion of the remote charge distribution. For large systems, the interaction between the quantum part and the classical part can be the most time-consuming contribution as there may be tens of thousands of multi-poles and localized polarizabilities, each of which has an interaction energy with the quantum system which contributes to the total energy.

These three calculations are the most time-consuming contributions in computational chemistry for biomolecular systems and have been parallelized in Dalton with MPI.

The Dalton program [5] is originally a Scandinavian collaboration that started out in the early 80's and now has developers from all Europe and a large worldwide user community (over 2000 licenses issued). It is primarily an electronic structure code aimed at solving the Schrödinger equation for the electrons of a molecular system at various levels of approximation. The methods involved are standard tools of quantum chemistry; Hartree-Fock (HF), density functional theory (DFT), multi-configuration self-consistent field (MCSCF), and coupled cluster (CC). These methods are available in other programs as well, but the area where Dalton specializes and has a leading position is molecular properties. These are calculated with the wave function as a starting point by employing perturbation theory, a.k.a. response theory. Molecular properties have a direct relation to an experiment, e.g. in spectroscopy, and is thus a more useful concept than an abstract wave function.

The quantum mechanics methods (QM) and quantum mechanics/molecular mechanics (QM/MM) included in Dalton allow users to study from small systems of a few atoms to huge molecular systems, such as proteins, cellular membranes and organic crystal surfaces. Examples include the prediction of the electronic and geometrical structure of proteins from first principles [6] or modelling optical probes inside proteins [7].

These computational chemistry techniques have improved dramatically over the last years along with the computer technology development. Thus, allowing researchers to solve problems unimaginable a couple of years ago. But we have to keep pushing the limits of chemistry computer applications such as Dalton, taking advantage of the current high performance infrastructures. We need optimized applications capable of scaling to thousands of cores, in that way, even bigger and more complex problems could be solved. Furthermore, with optimized and scalable applications, research results could be obtained faster, reducing the amount of money and time spent and shortening the distance between the lab and real end products.

In this paper we present a performance analysis and code optimization of Dalton as well as the performance results obtained after the optimizations.

The paper is structured as follows: Section II provides background information on the setup used for the experiments and the tools used for the analysis. In Section III and IV the overall architecture of the application and the results of our performance analysis are presented. Section V discusses the refactoring and optimizations applied and Section VI presents the results obtained. We close the paper in Section VII with concluding remarks and an outlook on future work.

II. Setup

The experiments were performed in a cluster with AMD Opteron 2374HE Quadcore processors at 2.2 GHz. Each machine node has 2 processors (8 cores) with 8 GB of DDR2 memory attached to each processor (for a total of 16 GB per node). The nodes are interconnected with an Infiniband network. The compiler used was the Intel compiler 11.1 and MPI library was OpenMPI 1.4.

The test case in this paper involves the calculation of the g-tensor of di-tert-butyl nitroxide solvated in water, a property that parameterizes the Zeeman effect in electron paramagnetic spectroscopy (EPR). The overall calculation is in two steps, first the wave function calculation and second, the property calculation (response theory). Each of the steps involves iterative methods and there are three contributions of the type listed above for each iteration. The runtime of the test case with 128 MPI processes in the initial program version lasted around 45 minutes.

The traces for the performance analysis have been generated with Extrae 2.1 and analysed with Paraver 3.99. Paraver and Extrae are part of the CEPBA-tools [8], an open source project developed by the Barcelona Supercomputing Center (BSC). Extrae is a set of libraries used to generate traces from different programming models like MPI, OpenMP, Pthreads, etc. It also allows to obtain hardware performance counters and call stack information from the execution. With this call stack information, the user can locate the different calls on the source code. Paraver is a powerful performance visualization and analysis tool based on traces. As the tool has no semantics, it can be used to analyze any information expressed on its input trace format. Different tracing packages can generate Paraver traces from different programming models or platforms.

As a difference with other tools, Paraver metrics are not hardwired but programmed. The tool offers a large set of time functions and a mechanism to combine two timelines. This approach allow that performance analyst experts have a huge flexibility when computing new metrics on the fly. As any set of views can be saved as a Paraver configuration file, non expert users can benefit from the tool basing his/her analysis on precomputed metrics defined by the experts.

Paraver provides two main types of display. The basic view is a timeline with a row per object. Code colours are used to drawn discrete metrics like user functions where each colour represents a value. A gradient scale (from light green to dark blue) is used for continuous metrics like hardware counters. The second type of view is the tables that are used to compute statistics like profiles, histograms and to correlate metrics. Both types of views can be displayed as a 2D array of pixels whose scalability is somehow similar to digital photos where image pixels are mapped to display window pixels.

III. Application architecture

Applications in the field of computational chemistry often have a task parallel structure. It supports the implementation of the irregular structure of quantum mechanical computations that vary in time and size due to varying base function sets for different kinds of atoms. Such algorithms have been implemented on the early supercomputers at best in a vectorized manner and with large shared memories. Implementations typically become more complex for clusters using message

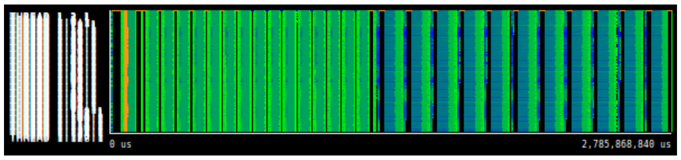


Fig. 1. View of the computation regions of the entire execution. X-axis is time and Y-axis processes. Colour ranges from green light meaning short computation to dark blue meaning long computation. Black is MPI operations.

passing techniques. The synchronization between processes often limits the scalability of the applications. [9]

Dalton's master-worker design pattern as base of its parallel implementation allows running the application with a very flexible number of nodes and achieves a good load-balance within the computational tasks. According to an input file describing the job the master process sends specific commands to the workers. These commands specify what kind of calculations have to be done. Furthermore, the master divides the computational tasks into smaller portions that are transfered to available workers to achieve good work balance.

A serious bottleneck of this architecture stems from the workload of the master. It has to receive the results of the computations from all workers as well as to compose them into the response function.

IV. Performance analysis

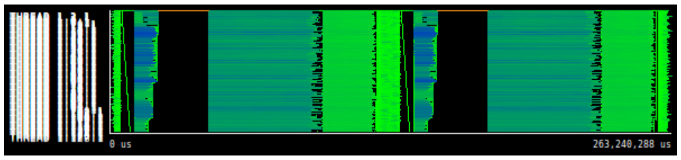
Fig. 1 shows the timeline of an entire simulation. This timeline has time on the X-axis and processes on the Y-axis. In this picture, we can see clearly two different patterns lasting around half of the execution each. The first pattern corresponds to the wave function calculation and the second one to the property calculation (response theory), as mentioned in section II. The colouring of the image shows the duration of the computation phases, meaning light green short computation and dark blue large computation. Black zones are MPI operations.

In this paper we will focus on the second phase of the execution, the determination of properties part. The first one, the wave function, remains as a future work.

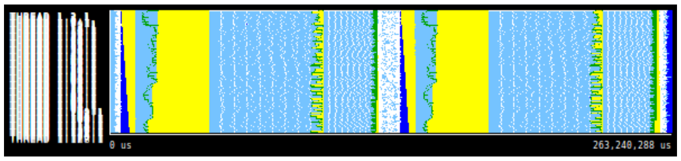
Fig. 2a shows the computational phases of two iterations from the properties part and Fig. 2b shows its corresponding MPI calls. As we can see from both figures, the iterations are highly dominated by MPI. The average of time in MPI is around 30% for the worker processes and 73% for the master as shown in Table I.

On the MPI timeline view (Fig. 2b), we can see some big yellow areas corresponding to worker processes waiting in MPI broadcast operations during a master-only sequential computation.

The master process, responsible of orchestrating the execution of the workers and computing results derived from their calculations, spends almost its 70% of execution time in MPI_Recv. We can see this behaviour closely in Fig. 3. While the master is doing some calculations, the workers are waiting in the long broadcast operation previously mentioned. Thereafter, the workers start to work and the master waits for their results.



(a) View of the computation regions of two iterations on the properties part. X-axis is time and Y-axis processes. Colour ranges from green light meaning short computation to dark blue meaning long computation. Black is MPI operations.



(b) MPI operations for two iterations on the properties part. Light blue is no MPI and all the other colours each different MPI operation. Yellow for MPI broadcast, dark blue for MPI send, white for MPI Receive and green for a Reduce operation.

Fig. 2. Computation and MPI calls of two iteration from the properties calculation phase

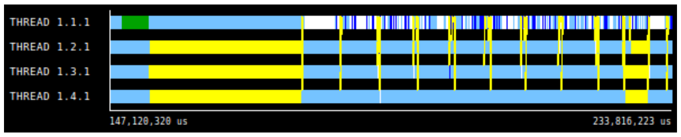


Fig. 3. View of the master/workers interaction: while the master is doing calculations, the workers wait in a MPI broadcast. After that, the workers compute and the master waits data from them. Colouring scheme of the picture: Yellow corresponds to a MPI_Bcast, white and dark blue to MPI_Recv and MPI_Send respectively. Light blue correspond to computation. The yellow lines between processes are MPI point to point communications.

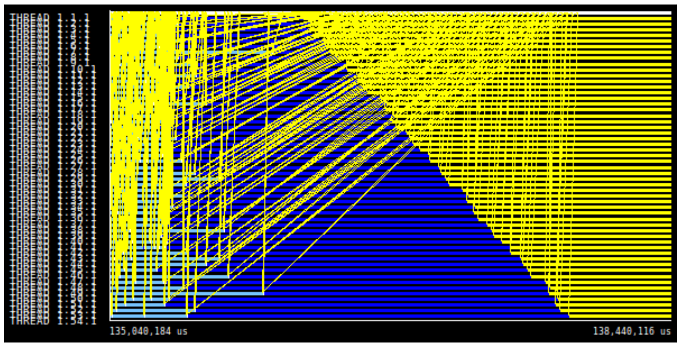


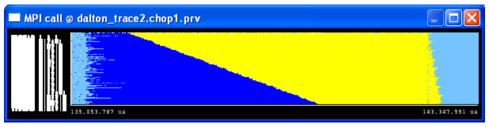
Fig. 4. View of a serialization phase in the execution. The master process is collecting information from the worker processes one by one in sequence.

There are other features related to the master/worker communication pattern as Fig. 4 shows. In this figure we can see a serialization phase were all the workers communicate in sequence with the master. There, the processes with bigger ranks have to wait until all point-to-point communications from the previous ranks have been completed. And after that, wait for a synchronization on a global operation.

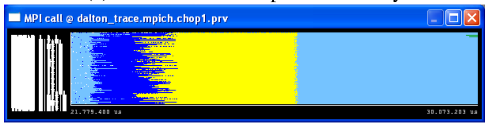
After further investigation, we discovered that this serialization was caused by the specific MPI library implementation used. In this case, OpenMPI is forcing the sequence of MPI receives in the master from 2 to N. In Fig. 5 we can see

TABLE I
TIME PERCENTAGES SPENT IN COMPUTATION AND IN MPI FOR THE MASTER PROCESS AND AN AVERAGE FOR ALL WORKER PROCESSES.

	Computation	MPI_Send	MPI_Recv	MPI_Bcast	MPI_Reduce
Master process	27.28%	0.11%	69.13 %	0.10 %	3.38 %
Workers Average	70.66%	2.89%	0.01%	24.69%	1.75%

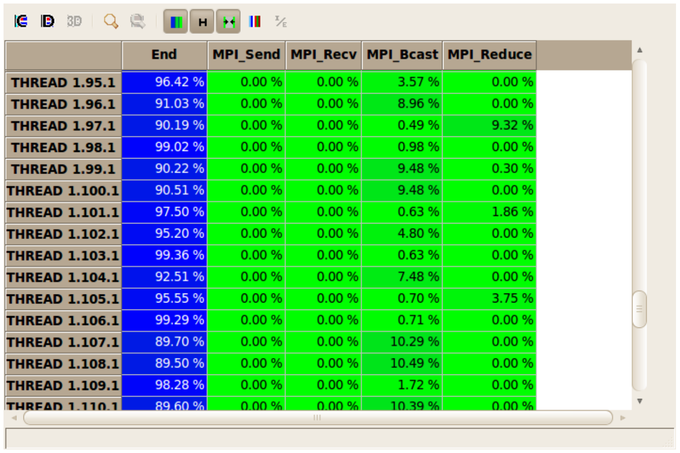


(a) Execution with OpenMPI library



(b) Execution with MVAPICH2 library

Fig. 5. Differences executing the same code with two different MPI libraries: OpenMPI and MVAPICH2. The dark blue color represents the MPI_Send from the workers and the yellow colour is a MPI broadcast.



	End	MPI_Send	MPI_Recv	MPI_Bcast	MPI_Reduce
THREAD 1.95.1	96.42 %	0.00 %	0.00 %	3.57 %	0.00 %
THREAD 1.96.1	91.03 %	0.00 %	0.00 %	8.96 %	0.00 %
THREAD 1.97.1	90.19 %	0.00 %	0.00 %	0.49 %	9.32 %
THREAD 1.98.1	99.02 %	0.00 %	0.00 %	0.98 %	0.00 %
THREAD 1.99.1	90.22 %	0.00 %	0.00 %	9.48 %	0.30 %
THREAD 1.100.1	90.51 %	0.00 %	0.00 %	9.48 %	0.00 %
THREAD 1.101.1	97.50 %	0.00 %	0.00 %	0.63 %	1.86 %
THREAD 1.102.1	95.20 %	0.00 %	0.00 %	4.80 %	0.00 %
THREAD 1.103.1	99.36 %	0.00 %	0.00 %	0.63 %	0.00 %
THREAD 1.104.1	92.51 %	0.00 %	0.00 %	7.48 %	0.00 %
THREAD 1.105.1	95.55 %	0.00 %	0.00 %	0.70 %	3.75 %
THREAD 1.106.1	99.29 %	0.00 %	0.00 %	0.71 %	0.00 %
THREAD 1.107.1	89.70 %	0.00 %	0.00 %	10.29 %	0.00 %
THREAD 1.108.1	89.50 %	0.00 %	0.00 %	10.49 %	0.00 %
THREAD 1.109.1	98.28 %	0.00 %	0.00 %	1.72 %	0.00 %
THREAD 1.110.1	89.60 %	0.00 %	0.00 %	10.39 %	0.00 %

Fig. 6. Percentages of time spent in computation and MPI for some workers doing QM/MM calculations. First columns shows that the computation for this zone is well balanced, with only aorund 10% of variation.

the differences executing the same code with OpenMPI and MVAPICH2. The timelines are synchronized in duration. We can see how just changing the MPI library used, the same piece of code runs 2x times faster and without any forced serialization.

After analysing the MPI phases, the computational parts should be analysed deeply. First, we start with the workload balance of the workers on each computational part. Fig. 6 shows the percentages spent in computation for an entire part of the code doing QM/MM calculations. As we can see, it is well balanced having at most around 10% of unbalance for some processes due to the irregular characteristics of the data partitioned. In this case, the data was divided in chunks 5% of the total data size, but this parameter is tuneable in the code and can be adjusted depending on the requirements of computation.

In order to analyze further the main computational regions we used the clustering tools [10] included in the CEPBA-tools suite. In Fig. 7 we can see a plot of the different computational regions clustered by their computational load (instructions) and their performance (IPC).

We can see that the main computation regions, colours light green and yellow in the Fig. 7b, have a poor IPC between 0.4 and 0.8. On the other hand, shorter regions in the timeline like dark green and red, report a better IPC but with a huge variability on their computational load (number of instructions executed).

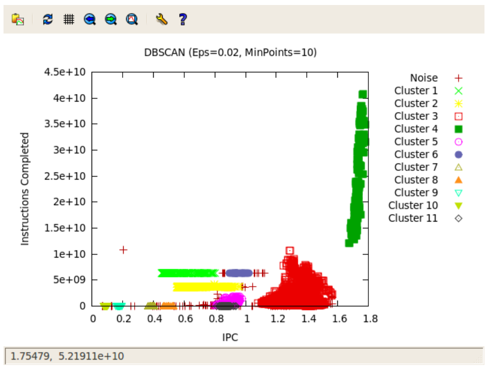
Clusters light green and yellow are well balanced regarding their number of instructions but they show some variablity on their IPC. Actually we can see on Fig. 7a how the variability in Cluster 1 continues in a separate cluster. Taking a look at the mapping of clusters in Fig. 7b we see that Cluster 6 corresponds to the last unbalanced chunks of the application mapped to Cluster 1. Further analysis with hardware counters showed that the low IPC in these clusters is due to the big number of expensive floating-point operations like divisions or square roots.

V. OPTIMIZATIONS

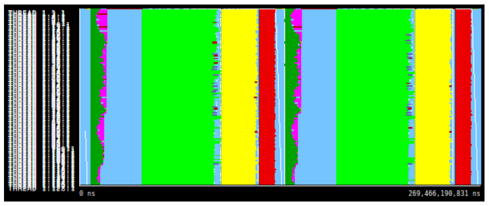
The performance analysis done in the previous section shows that the current main bottleneck in the scaling for Dalton consists in its master-worker design. It is of utmost importance to reduce the time spent in the master sequential phases as well as make it lighter, from the workload point of view.

The optimizations in the application code also focused on the part for the calculation of the molecular properties, that is, the calculation of the response function. These function values consist of several contributions from different modelling approaches like quantum mechanics or density functional theory.

The bottleneck in the communication between master and worker processes has been removed through the introduction of a ”team of masters”. The different contributions to the response function can be calculated independent and in parallel. All workers took part in the computation of every contribution before the optimization. Now after the refactoring, the available set of workers is partitioned into groups that have the task to compute only one contribution as shown in Fig. 8. Every group has a size adapted to the computational work needed for the calculation and its own master that collects the results. A substantial ease of the master's workload can be reached in that way. Smaller worker groups dealing with only one task lead to shorter waiting times for point to point communications as well as less complex and shorter collective communications. Furthermore, multiple workers can communicate at the same time realizing better



(a) Plot of the computation bursts clusterized by intructions and IPC.



(b) Generated clusters mapped on the trace.

Fig. 7. Clustering of the computational bursts. Both pictures share the same colouring scheme.

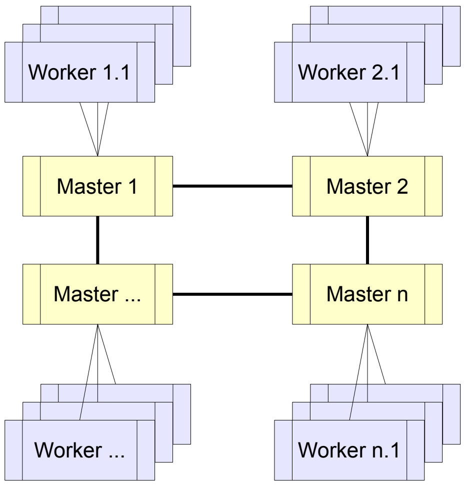
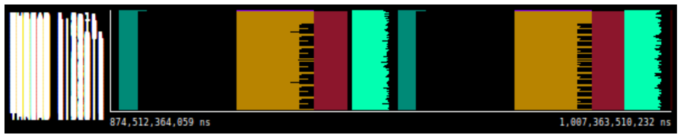


Fig. 8. New structure of the application with workers grouped in teams with their own team master.

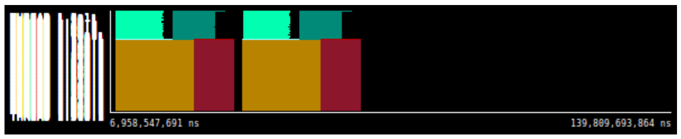
use of the interconnect and a higher degree of parallelism. The collected contributions are finally exchanged between the masters, which also compute the response function and initiate a new iteration until the convergence is reached.

The initial performance analysis showed computational functions with a low density of floating-point instructions. One such function is the routine that adds the contribution from the DFT computations to the orbitals. This routine caused the long phase when all workers had to wait for the sequential computation of the master process. This is a typical example for the specifics of applications in computational chemistry too. The preparation of the matrices is often a time consuming work while the linear algebra operations itself can be solved very efficiently by optimized libraries. It was possible to reach a substantial improvement in the single node performance of this matrix composition through typical loop optimizations in the master sequential phase in our case. [11]

Other improvements have been introduced in the code. For example, the communication phases with point-to-point operations serialized by the MPI library as seen in Fig. 5



(a) Initial version



(b) Optimized version

Fig. 9. The coloured blocks show the calculation of the different contributions and how they are distributed among the workers during two iterations of the simulation: her_nodstr, dft_lin_respab_b, qmmmlno_s1 and qmmmlno_s2. Both pictures are synchronized in duration. We can see how the optimized version takes less than half of the time of the initial version.

have been changed. This communication between workers and master is driven now by MPI collectives. In this way, we reduce the number of point-to-point communications, we increase scalability of the code and we are less dependent to the MPI library implementation to obtain better performance.

VI. Results

The performance of the properties phase for both codes (the original and the optimized one) have been measured. The test case is the calculation of the g-tensor of di-tert-butyl nitroxide solvated in water that combines contributions from QM/MM and DFT calculations. Its strong scalability behaviour has been tested with runs from 128 to 1024 MPI-processes. Speedups and parallel efficiency are derived from the 128 MPI processes run as base and calculated as (1).

$$S_p = \frac{T_{128}}{T_p} \qquad E_p = \frac{S_p}{p} \tag{1}$$

The optimized program version executes the QM/MM and DFT calculations in parallel as seen in Fig. 9. It also uses the optimized matrix composition routine. The latter saves 3 minutes of the runtime. Those are dependent on the number of cores between 46% and 63% of the improvement. This

TABLE II
TIMES, SPEEDUPS AND EFFICIENCY FOR THE PROPERTIES PART IN DALTON.

Num. Processes	Time		Speed Up		Efficiency	
	Initial version	Optimized version	Initial version	Optimized version	Initial version	Optimized version
128	24m 03s	19m 19s	1	1	1	1
256	16m 22s	10m 40s	1.47	1.81	0.74	0.88
512	12m 33s	06m 45s	1.91	2.86	0.48	0.75
1024	11m 30s	05m 00s	2.01	3.86	0.26	0.47

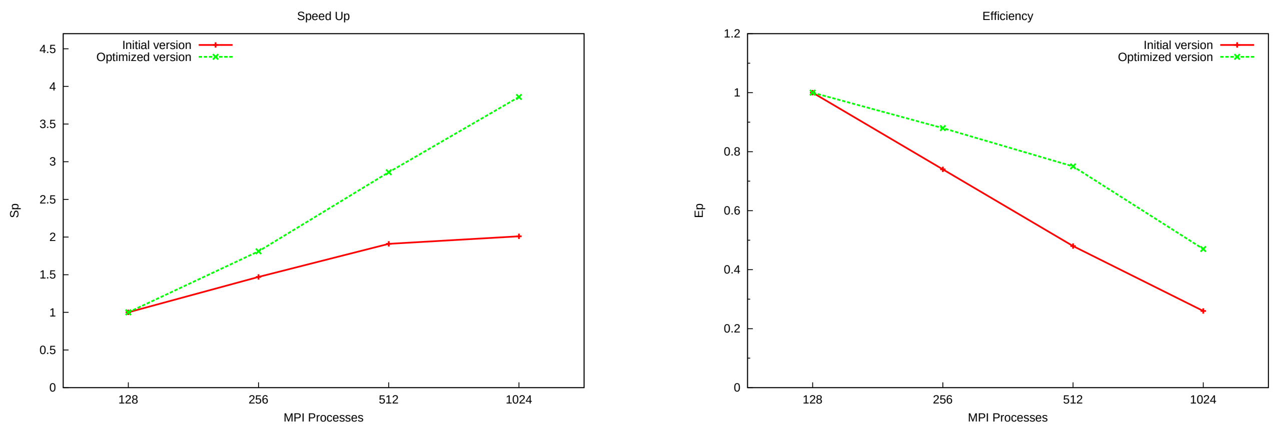


Fig. 10. Speedup and efficiency for the initial and for the optimized version

improvement is especially valuable to increase the efficiency because the calculations executed here are in a sequential phase of the algorithm letting all workers in an idle status. The remaining improvement is gained from the parallel calculation of the different contributions to the response function.

The original program version scales until 512 cores (see Table II) and provides so far a parallel efficiency around 50% (see Fig. 10). Almost no speedup can be gained beyond this number of nodes. Whereas the optimized version run now up to 1024 cores providing so far a parallel efficiency around 50% or more.

We can look now again at the percentages of computation and MPI time in our current version and compare them to the ones from Table I in section V. We see that our optimized version has increase its average computation time from initial value of 70% for workers to 86%, reducing its average MPI time from around 30%to 14%. This implies that now the application does a better use of the resources and wastes less time in synchronization or communication.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we have shown a characterization and optimization of the properties calculation phase of Dalton. First, we identified and explained several MPI related aspects from the application that could be improved. The master-worker design has been discussed as well as some master serial phases. We also saw how some parts of the application behaved completely different depending on the MPI library used. Afterwards, we analysed the workload among the workers in the computation and characterize these computational parts regarding their IPC and computational load (number of instructions).

Guided by all these analyses, we improved the single-node performance of the master to shorten serial phases needed by the current algorithm. Furthermore, the structure of parallel tasks has been refactored allowing now the independent parallel computation of the contributions to the response function. These improvements lead to the result that the computational efficiency has been increased as well as that parallel runs of Dalton may use now a larger number of processors and provide higher speedups than before.

Our promising results demonstrate us the potential for further optimizations. The number of worker processes calculating the different contributions is now fixed and based on previous exemplary simulations. We will implement a dynamic load balancing scheme that distributes the worker processes on the basis of a realtime monitoring of application performance and idle times. The user could define tasks, matching each contribution in this case, and the system will execute them automatically in the best way to maximize performance. We already have a first prototype of this runtime system [12] and we will start testing it with Dalton in a near future.

To reduce the number of communications between master and workers the use of hybrid parallelization techniques can be considered. This would lead to less communication operations with larger messages as well as more flexibility for load balancing between the cores on one cluster node. Deeper studies of the computational zones with low IPC could be done in order to optimize them and speed up more the computation. Some other techniques like the use of accelerators could be tested. Our first analyses of these zones with low IPC showed us that these phases have lots of costly floating-point operations like divisions and square roots. Because of that, they are really good candidates to be executed in accelerators like GPUs. Beyond optimizations of existing algorithms we would like to research further on algorithms that can calculate the response function in a distributed manner not requiring the transfer of all contributions to one or several master processes. Finally, the same analysis and enhancement of code that has been done here needs to be done on the wave function phase.

ACKNOWLEDGMENT

This work was partly funded by the European Commission (contract INFSO RI-261523) through the ScalaLife Project [13] and the Swedish e-Science Research Centre (SeRC) [14].

REFERENCES

[1] J. Almlöf, J. K. Faegri, and K. Korsell, *J. Comp. Chem.*, vol. 3, p. 385, 1982.
[2] W. Kohn and L. J. Sham, *Phys. Rev.*, vol. 140, p. A1133, 1965.
[3] P. Hohenberg and W. Kohn, *Phys. Rev.*, vol. 136, p. B864, 1964.
[4] A. Warshel, *J. Mol. Biol.*, vol. 103, p. 227, 1976.
[5] *Dalton, a molecular electronic structure program, Release DALTON2011*, 2011, http://daltonprogram.org/.
[6] S. Reine, A. Krapp, M. F. Iozzi, V. rn Bakken, T. Helgaker, F. P. owski, and P. S. ek, "An efficient density-functional-theory force evaluation for large molecular systems," *The Journal of Chemical Physics*, vol. 133, no. 4, p. 044102, 2010. [Online]. Available: http://link.aip.org/link/?JCP/133/044102/1
[7] N. Murugan, J. Kongsted, Z. Rinkevicius, and H. A. gren, "Structure and color modeling of optical probes within proteins. a case study of nile red in beta-lacto globulin," *The Journal of Chemical Physics*, submitted to publication.
[8] Cepba-tools team @ bsc. [Online]. Available: http://www.bsc.es/plantillaF.php?cat_id=52
[9] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, Eds., *Sourcebook of parallel computing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.
[10] J. Gonzalez, J. Gimenez, and J. Labarta, "Automatic detection of parallel applications computation phases," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, may 2009, pp. 1 –11.
[11] J. Levesque and G. Wagenbreth, *High Performance Computing: Programming and Applications*, ser. Chapman & Hall/CRC Computational Science. CRC Press Taylor & Francis Group, 2011.
[12] M. Schliephake, X. Aguilar, and E. Laure, "Design andimplementation of a runtime system for parallel numerical simulations on large-scale clusters," *Procedia Computer Science*, vol. 4, no. 0, pp. 2105 – 2114, 2011, ¡ce:title¿Proceedings of the International Conference on Computational Science, ICCS 2011¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050911002882
[13] Scalalife project. [Online]. Available: http://www.scalalife.eu
[14] The swedish e-science research center (serc). [Online]. Available: http://www.e-science.se