Scheduling Parallel Eigenvalue Computations in a Quantum Chemistry Code

Martin Roderus[1], Anca Berariu[1], Hans-Joachim Bungartz[1], Sven Krüger[2], Alexei Matveev[2], and Notker Rösch[2]

[1] Institut für Informatik, Technische Universität München, Germany
[2] Department Chemie and Catalysis Research Center, Technische Universität München, Germany

Abstract. The application of High Performance Computing to Quantum Chemical (QC) calculations faces many challenges. A central step is the solution of the generalized eigenvalue problem of a Hamilton matrix. Although in many cases its execution time is small relative to other numerical tasks, its complexity of $\mathcal{O}(N^3)$ is higher, thus more significant in larger applications. For parallel QC codes, it therefore is advantageous to have a scalable solver for this step.

We investigate the case where the symmetry of a molecule leads to a block-diagonal matrix structure, which complicates an efficient use of available parallel eigensolvers. We present a technique which employs a malleable parallel task scheduling (MPTS) algorithm to schedule instances of sequential and parallel eigensolver routines from LAPACK and ScaLAPACK. In this way, an efficient use of hardware resources is guaranteed while overall scalability is facilitated. Finally, we evaluate the proposed technique for electronic structure calculations of real chemical systems. For the systems considered, the performance was improved by factors of up to 8.4, compared to the previously used, non-malleable parallel scheduling approach.

1 Introduction

We report on the development and the implementation of a task scheduling algorithm for the parallel computation of the generalized eigenvalue problem as a central task of electronic structure calculations in quantum chemistry, exemplified for the case of the Kohn-Sham formulation of density functional theory [17] as implemented in the program ParaGauss [3,4,11]. Here the Hamilton matrix H represents the Kohn-Sham Hamilton operator of the electronic structure problem. The molecular orbitals are expanded in a finite set of atom-centered Gaussian-type functions, multiplied by spherical harmonics, to describe the angular variation of atomic orbitals. The eigenvalues of H are the energies of the molecular orbitals and the eigenvectors are the coefficients of orbitals with respect to the "basis set" of Gaussian-type functions. This eigenvalue problem has the following generalized form:

$$\sum_j H_{ij} c_{jk} = e_k \sum_j S_{ij} c_{jk}, \tag{1}$$

where H is the Hamilton matrix, and the overlap matrix S represents the metric of the nonorthogonal basis set, e_k are the orbital energies, and c_k the eigenvectors. H is a symmetric real and dense matrix for most applications. For molecules with a closed-shell electronic structure, in general, a single matrix has to be diagonalized; for open-shell systems, two such eigenvalue problems of the same size have to be solved. A more complicated situation arises when the spatial symmetry of a molecule is exploited to simplify the calculation. Transformation of the basis into a symmetry adapted form, according to the irreducible representations of the point group of the molecule, leads to the same block-diagonal structure of H and S. Cutting down significantly the computational effort of an electronic structure calculation, the symmetrization also implies a possibility to introduce parallelism to the diagonalization step [3]: the sub-matrices are sorted by their size in descending order; whenever a processor becomes available, it is employed to perform a sequential diagonalization of the first sub-matrix in the list. However, this approach, also known as *LPT algorithm* [14] as a special form of *list scheduling* [13], shows very limited scalability: the sequential execution time for the largest matrix is a lower bound on the overall execution time, and the number of matrices is an upper bound on the number of processors which can be used; see Fig. 1a.

While the majority of current quantum chemistry problems are solved without invoking spatial symmetry constraints, exploiting point group symmetry can offer significant advantages in the field of nano particles. Clusters of various materials in the size range of nano particles ($1 - 10\,\mathrm{nm}$ diameter) typically contain about 50 to several thousand atoms. Molecules with more than hundred heavy atoms still represent a challenge to accurate quantum chemistry methods and symmetry thus yields a convenient way of applying these methods to nano sized model species in the relevant size range. Moreover, highly symmetric metal nano particles have also been shown to be useful as versatile models of transition metal particles that act as catalysts of chemical reactions [26,25]. For this class of very large molecular systems, point groups with a small set of irreducible representations are invoked to describe several instances of a reaction, on symmetry equivalent facets of a larger (symmetric) metal particle. Such nano particles represent prototypical applications, where a generalized matrix eigenvalue problem decomposed into several sub-matrices, typically 4–10, of different sizes has to be solved.

For a typical application, the dimension N of the basis set, which corresponds to the size of H, ranges from 500 up to several thousand. The corresponding generalized eigenvalue problem represents overall a smaller task of the electronic structure calculation, but it is a step growing with $\mathcal{O}(N^3)$ and thus becomes more important with increasing problem size. On the other hand, the solution of (1) is part of the Self-Consistent Field (SCF) method, an iterative scheme. A typical SCF electronic structure determination demands $30 - 100$ (or more) iterations, and the optimization of the structure of a larger system requires 50 to several hundred electronic structure determinations. This adds up to $10^3 - 10^5$ diagonalizations in a single typical application. Thus, an efficient parallel algorithm for the simultaneous diagonalization of several symmetric matrices is a worthwhile goal when optimizing a parallel quantum chemistry code for large scale and high performance applications.

There have been approaches to include ScaLAPACK [6] as parallel eigensolver into electronic structure calculations, see [15,24]. However, in these calculations, point group symmetry was not exploited, so H was a single dense, symmetric matrix which was diagonalized by one instance of the solver. In our case, the diagonalization of $H \in \mathbb{R}^{N \times N}$ can be divided into several smaller subproblems, which significantly reduces the complexity of $\mathcal{O}(N^3)$. Thus, to benefit from this computational advantages, a different parallelization strategy is required.

In this paper, we propose the following novel approach: the sequential and parallel routines `DSYGV` and `PDSYGV` from LAPACK [2] and ScaLAPACK [6], respectively, are employed to diagonalize the set of sub-matrices independently. A malleable task scheduling algorithm allots to each matrix a processor count and schedules the instances of the eigensolvers to achieve proper load balancing and thus reduce the overall execution time.

The rest of the paper is organized as follows: Section 2 provides an abstract formulation of the problem. Section 3 gives a short overview over malleable parallel task scheduling and describes the algorithm used. Section 4 presents a technique to construct a cost function required by the scheduling algorithm. In Sect. 5, the technique is evaluated on chemical systems of practical relevance. Finally, we offer some conclusions from this work in Sect. 6.

2 Description of the Problem

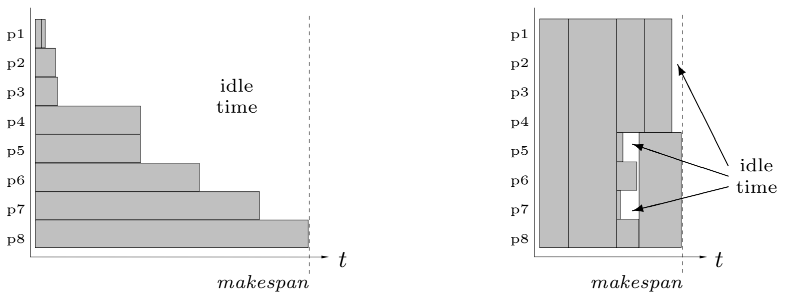
The objective is to find all eigenvalues and eigenvectors of a set of symmetric matrices of different size. The term "task" will be used as synonym for a single matrix diagonalization which is processed by an eigensolver.

Hence, the problem can be formulated as follows: given is a set of n tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$ and a set of m identical processors. Each task is assigned a matrix of defined size, so there is a set of sizes $\mathcal{S} = \{S_1, \ldots S_n\}$ with the relation $T_i \mapsto S_i$. The matrix sizes in $\mathcal{S}$ can vary arbitrarily. The tasks are independent and nonpreemptable. Furthermore they are malleable, i.e., a task may be executed by an arbitrary number of processors $p \leq m$, resulting in different execution times. There is a *cost function* which models this behavior. As the employed routine – the eigensolver – is identical for each task, there is only one cost function denoted as

$$t : (S_i, p) \mapsto t_{i,p}, \tag{2}$$

which predicts the execution time of task T_i with size S_i when executed on p processors.

Furthermore, there is a set of processor counts $\mathcal{P}$, where its elements P represent a possible number of processors on which a task can be executed. The *makespan* d is the overall time required to process all tasks from $\mathcal{T}$. The goal is



(a) LPT Scheduling (b) Malleable Task Scheduling

Fig. 1. Example of a scheduling of 9 tasks on 8 processors. The grey boxes represent single tasks. Their widths indicate the execution time and their heights the number of processors they are executed on.

to find a scheduling with a minimal makespan. As the problem arises in many practical applications, it is well studied and known as *malleable parallel task scheduling* (MPTS). An example is depicted in Fig. 1b.

3 Malleable Parallel Task Scheduling

3.1 Related Work

The MPTS introduced in Sect. 2 is a common scheduling problem and has been subject of frequent discussion over the last decades, see [7,8,19]. MPTS is a generalization of a sequential scheduling problem, which is NP-complete in the strong sense [12]. Therefore, a variety of approximation algorithms with polynomial runtime exist. A common approach is based on a two-phase strategy, first introduced by Turek et al. [23]. The idea is to find a processor allotment for each task in the first step and to solve the resulting nonmalleable parallel task scheduling (NPTS) problem in the second step. Ludwig and Tiwari [18] suggested such an algorithm with approximation factor 2, which shall be the basis of the strategy proposed here. Mounié et al. [20] followed a different approach by formulating a Knapsack problem as core component. They provide the currently best practical algorithm with approximation factor $\frac{3}{2} + \epsilon$ for any fixed $\epsilon > 0$. When the tasks have to be executed on processors with successive indices, Steinberg [22] proposed an adapted strip-packing algorithm with approximation factor 2. Furthermore, Jansen [16] gave an approximation scheme with makespan at most $1 + \epsilon$ for any fixed $\epsilon > 0$. There exist other algorithms for special cases of the MPTS with approximation factors close to one (e.g. [9] for identical malleable tasks), but those do not apply for our case.

The problem stated in this paper is a standard MPTS problem, so the algorithms mentioned above could in principle be applied. However, as we will see, the number of sub-matrices to be processed is limited for most cases. This allows us to modify the algorithm from [18] by introducing a combinatorial approach in order to find a better solution.

3.2 The Algorithm

We focus on a two-phase approach. In the first phase, a number of allotted processors for each task is determined. This step performs the transformation from an MPTS to an NPTS. In the second phase, an optimal scheduling for the nonmalleable tasks is constructed.

Processor Allotment: In [18], Ludwig and Tiwari showed how a processor allotment can be found in runtime $\mathcal{O}(mn)$. The algorithm also computes a lower bound ω on the optimal makespan d^* such that $\omega \leq d^* \leq 2\omega$. The basic idea is to find a number of allotted processors $P_{T_i} \in \mathcal{P}$ for each task T_i such that $t_{i,P_{T_i}} \leq \tau$. $\tau \in \mathrm{I\!R}$ is defined as an upper bound for the execution time of each task, and P_{T_i} the minimum number of processors which are required to satisfy this condition. The goal is to find the minimum τ^* which produces a feasible allotment for each task. Furthermore, the values of τ to be considered can be limited to a certain set $\mathcal{X} = \{t_{i,P_j} : i = 1 \ldots n, j = 1 \ldots m\}$. Thus, $|\mathcal{X}| = mn$. Once τ^* has been found, the algorithm yields an allotment, which will subsequently be used for solving the NPTS problem. The algorithm requires t to be *strictly monotonic*: $t_{i,p_1} > t_{i,p_2}$ for $p_1 < p_2$. This property is not generally given, but can be achieved by a suitable choice of the cost function (see below). For further details, please refer to the original paper [18].

Solution of NPTS: Ludwig and Tiwari presented a technique which "*… takes any existing approximation algorithm for NPTS and uses it to obtain an algorithm for MPTS with the same approximation factor.*"[18]. In other words, if there is a way to find an optimal schedule for the NPTS, it simultaneously yields an optimal solution for the MPTS. For each irreducible representation, one block on the diagonal of the Hamilton matrix H has to be diagonalized; in most applications, the number of such blocks is limited by 10. This includes the important point groups I_h and O_h and their subgroups as well as all point groups with up to fivefold rotational axes as symmetry elements [1]. Thus, point groups including more than 10 irreducible representations are very rarely encountered in practical applications. For such a limited problem size, we can achieve an optimal solution by a combinatorial approach. The number of possible permutations σ of an NPTS is $n!$; in our case, this leads to a maximum number of $10! \approx 3.6 \cdot 10^6$.

Algorithm 1 shows the routine to find the scheduling with the minimum makespan d^*. A scheduling is represented by a sequence of task numbers σ, from which a scheduling is generated in Algorithm 2. For simplification, we do not explicitly provide the routine that generates the permutations.

Algorithm 1. Finds the scheduling sequence σ^* from which MakeSchedule (algorithm 2) generates a scheduling with the minimum makespan

```
1. Find the tasks which have been allotted all processors: $\mathcal{T}_m \subseteq \mathcal{T} = \{T_i : P_{T_i} = m\}$
2. Schedule $\mathcal{T}_m$ at the beginning
3. For each possible permutation $\sigma$ of $\mathcal{T} \setminus \mathcal{T}_m$
   (a) Call MakeSchedule to generate a scheduling from $\sigma$
   (b) $\sigma^* \leftarrow \sigma$ if makespan of $\sigma <$ makespan of $\sigma^*$
```

Algorithm 2. The procedure to generate a scheduling from a scheduling sequence σ. It allots to each task $\sigma(i) \mapsto T_i$ a start time and an end time ($\sigma(i)$.startTime and $\sigma(i)$.endTime, respectively), as well as a set of allotted processors ($\sigma(i)$.allottedProcessors). There is a set of processors, $\mathcal{C} = \{C_1 \dots C_m\}$; each element C_i has an attribute C_i.availableTime, which points out until which time the processor is occupied and, thus, the earliest time from which on it can be used for a new task.

```
procedure MakeSchedule($\sigma$)
    for all $C_i \in \mathcal{C}$ do
        $C_i$.availableTime $\leftarrow 0$
    end for
    for $i \leftarrow 1, |\sigma|$ do
        $\mathcal{C}_{\text{tmp}} \leftarrow$ first $P_{T_i}$ processors which are available
        $\sigma(i)$.startTime $\leftarrow$ max(available times from $\mathcal{C}_{\text{tmp}}$)
        $\sigma(i)$.endTime $\leftarrow \sigma(i)$.startTime $+ t_{i,P_{T_i}}$
        $\sigma(i)$.allottedProcessors $\leftarrow \mathcal{C}_{\text{tmp}}$
        for all $C_j \in \mathcal{C}_{\text{tmp}}$ do
            $C_j$.availableTime $= \sigma(i)$.endTime
        end for
    end for
end procedure
```

4 Cost Function

The scheduling algorithm described requires a cost function which estimates the execution time of the (Sca)LAPACK routines `DSYGV` and `PDSYGV`. It is difficult to determine upfront how accurate the estimates have to be. However, the validation of the algorithm will show whether the error bounds are tight enough for the algorithm to work in practice.

The ScaLAPACK User's Guide [6] proposes a general performance model, which depends on machine-dependent parameters such as floating point or network performance, and data and routine dependent parameters such as total FLOP or communication count. In [10], Demmel and Stanley used this approach to evaluate the general performance behavior of the ScaLAPACK routine `PDSYEVX`. The validation of the models shows that the prediction error usually lies between 10 and 30%. Apart from that, for the practical use in a scheduling algorithm, it exhibits an important drawback: to establish a model of that kind, good knowledge of the routine used is required. Furthermore, each routine needs its own model; thus, if the routine changes (e.g. due to a revision or the use of a different library), the model has to be adapted as well.

Here we follow a different approach: the routine is handled as a "black box". Predictions of its execution time are based on empirical data, which are recorded by test runs with a set of randomly generated matrices on a set of possible processor allotments $\mathcal{P}$. Then, with a one-dimensional curve-fitting algorithm, a continuous cost function t is generated for each element of $\mathcal{P}$. Thus, each $P \in \mathcal{P}$ has a related cost function $t_P : S \mapsto t_{P,S}$ (do not confuse with (2)).

ScaLAPACK uses a two-dimensional block cyclic data distribution. For each instance of a routine, a $P_r \times P_c$ process grid has to be allocated with P_r process rows and P_c process columns. However, the ScaLAPACK User's Guide [6] suggests to use a square grid ($P_r = P_c = \lfloor\sqrt{P}\rfloor$) for $P \geq 9$ and a one-dimensional grid ($P_r = 1; P_c = P$) for $P < 9$. Following this suggestion results in a reduced set of processor configurations, e.g. $\mathcal{P} = \{1, 2, \ldots, 8, 9, 16, 25, \ldots, \lfloor\sqrt{m}\rfloor^2\}$, which will be invoked here.

We used the method of least squares to fit the data. For our purposes, it shows two beneficial properties:

- the data are fitted by polynomials. These are easy to handle and allow us to generate an estimated execution time $t_{P,S}$ with low computational effort;
- during the data generation, processing of a matrix sometimes takes longer than expected due to hardware delays (see circular marks in Fig. 2). As long as those cases are rare, their influence on the cost function is minor and can be neglected.

Finally, we combine the emerging set of P-related cost functions to form the general cost function (2). However, in practice, when a certain number of allotted processors is exceeded, parallel routines no longer feature a speedup or even slow down, see [24]. This behavior does not comply with the assumption of a general monotonic cost function. To satisfy this constraint, we define (2) as follows:

$$t_{i,p} = \min_{P}\{t_{P,S_i} : P \in \mathcal{P} \wedge P \leq p\}. \tag{3}$$

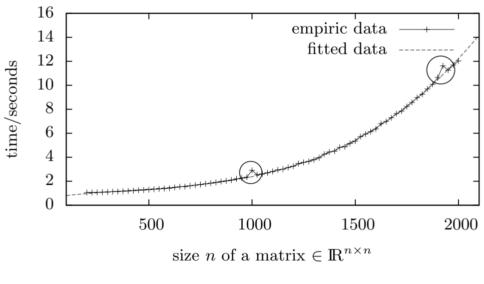


Fig. 2. Execution time measurements of the routine PDSYGV diagonalizing randomly generated matrices. The curve labeled "fitted data" represents a polynomial of degree 3, which was generated by the method of least squares from the empiric data.

All possible processor counts $P \in \mathcal{P}$ are considered which are smaller than or equal to p. The P which results in the smallest execution time for the given S also determines the P-related cost function and thus the result t of the general cost function (3).

5 Evaluation

We evaluated the presented scheduler for two molecular systems as example applications: the gold cluster compound $Au_{55}(PH_3)_{12}$ in symmetry S_6 and the palladium cluster Pd_{344} in symmetry O_h. Table 1 lists the sizes of the symmetry adapted basis sets which result in $\mathcal{S}_{Au_{55}}$ and $\mathcal{S}_{Pd_{344}}$.

Table 1. The resulting point group classes (PGC) of the two example systems $Au_{55}(PH_3)_{12}$ in symmetry S_6 and Pd_{344} in symmetry O_h. The third row contains the sizes n of matrices $\in \mathbb{R}^{n \times n}$ which comprise the elements of $\mathcal{S}_{Au_{55}}$ and $\mathcal{S}_{Pd_{344}}$.

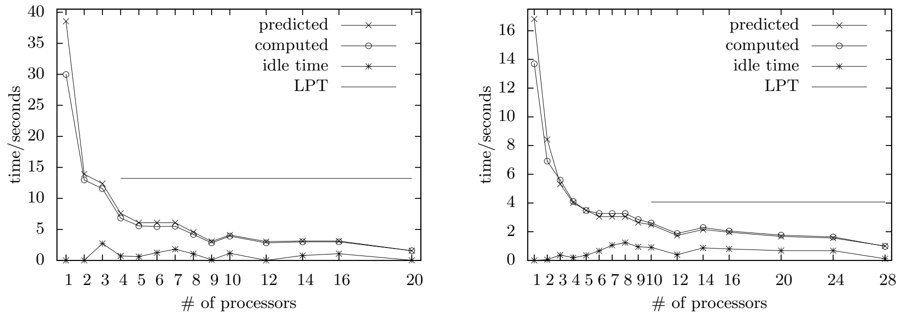
	$Au_{55}(PH_3)_{12}$				Pd_{344}									
i	1	2	3	4	1	2	3	4	5	6	7	8	9	10
PGC	A_g	E_g	A_u	E_u	A_{1g}	A_{2g}	E_g	T_{1g}	T_{2g}	A_{1u}	A_{2u}	E_u	T_{1u}	T_{2u}
S_i	782	1556	782	1560	199	317	513	838	956	1110	317	471	785	956

Test platform was an SGI Altix 4700, installed at the Leibniz Rechenzentrum München, Germany. The system uses Intel Itanium2 Montecito Dual Cores as CPUs and has an SGI NUMAlink 4 as underlying network. As numerical library, SGI's SCSL was used to provide BLAS, LAPACK, BLACS and ScaLAPACK support. For further details on the hard- and software specification, please refer to [21].

We performed time and load measurements using the VampirTrace profiling tool [5]. For that purpose, we manually instrumented the code, inserting calls to the VT library. As a result, the two relevant elements of the scheduling algorithm could be measured separately: the execution time of the eigensolvers and the idle time (see below).

Two negative influences on the parallel efficiency can be expected: firstly, as virtually every parallel numerical library, the performance of ScaLAPACK does not scale ideally with the number of processors. Consequently, the overall efficiency worsens the more the scheduler parallelizes the given tasks. The second negative influence arises from the scheduling itself. As Fig. 1 shows, a processor can stay idle while waiting for other processors or the whole routine to finish. Those idle times result in load imbalance, hence a decreased efficiency.

Figure 3 shows the execution times of a scheduled diagonalization during one SCF cycle. The gap between the predicted and computed execution time is reasonably low (at most $\approx 10\%$) except for the case when $p = 1$. This shows that the cost function, which generates the values for the predicted makespan, works sufficiently accurate to facilitate the practical use of the scheduling algorithm.



(a) $Au_{55}(PH_3)_{12}$ (b) Pd_{344}

Fig. 3. Time diagrams of the two test systems $Au_{55}(PH_3)_{12}$ and Pd_{344}. Considered are the execution times of the diagonalization module during one SCF iteration. The curves labeled "predicted" show the predicted makespan of the scheduling algorithm, whereas the curves labeled "computed" provide the real execution time of the scheduled eigensolvers. The curves labeled "idle time" represent the average time, during which a processor was not computing. The lines labeled "LPT" indicate the execution time of the sequential LAPACK routine computing the largest matrix from $\mathcal{S}$ and yield thus the best possible performance of the previously used LPT-scheduler (see Sect. 1).

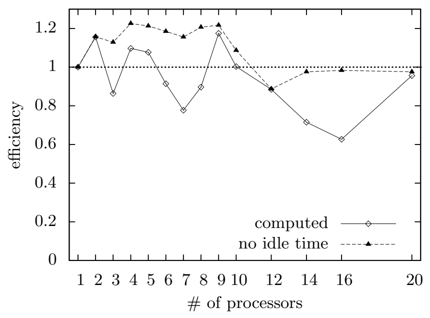
The figure also shows a lower bound on the execution time of a sequential scheduler ("LPT"-line). To recapitulate the basic idea of the previously used LPT-scheduler: all matrices are sorted by their size and accordingly scheduled on any processor which becomes available. There the matrix is diagonalized by a sequential LAPACK eigensolver routine (see Fig. 1a). However, this performance bound is now broken and the execution time is improved beyond this barrier by our new algorithm.

The diagonalization of the first system, $Au_{55}(PH_3)_{12}$, scales up to 20 processors, whereas the LPT-scheduler can only exploit up to 4 processors. The proposed MPTS scheduler is faster by a factor of about 2 when using 4 processors and by a factor of 8.4 when using 20 processors.

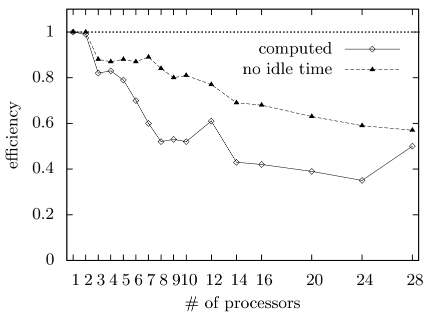
For the diagonalization of the second system, Pd_{344}, the execution time improved for up to 28 processors. Compared to the LPT-scheduler, which in this case can only exploit up to 10 processors, the MPTS-scheduler is faster by the factor of 1.6 when using this processor number and about 4 when using 28 processors.

The parallel efficiency is given in Fig. 4. For the system $Au_{55}(PH_3)_{12}$, a superlinear speedup was achieved for the cases $p = \{2, 4, 5, 9\}$. One can also see for both examples that the idle times of the scheduling can cause a notable loss of efficiency. Further investigations on the scheduling algorithm to reduce those time gaps would thus be an opportunity to improve the overall efficiency.

One also has to consider the cost of establishing the scheduling. It is difficult to estimate this cost beforehand, but recall that the scheduling, once computed, can be re-used in each recurring SCF cycle, at least 10^3 in a typical application



(a) $Au_{55}(PH_3)_{12}$



(b) Pd_{344}

Fig. 4. The parallel efficiency in computing the two test systems $Au_{55}(PH_3)_{12}$ and Pd_{344}. The curves labeled "computed" show the efficiency according to the execution times displayed in Fig. 3. The curve labeled "no idle time" represents the efficiency, where the idle-times of the scheduling are subtracted from the execution times. It thus indicates the parallel efficiency of the ScaLAPACK-routine `PDSYEVX` as used in the given schedules.

(see Sect. 1). Thus, a reasonable criterion is the number of SCF cycles required to amortize these costs. In the worst case considered, for 10! possible task permutations (see Sect. 3.2), our implementation requires a runtime of ≈ 34 s. This case has to be considered for the second test system Pd_{344}, as its symmetry group O_h implies a total number of 10 matrices. In an example case, where the these matrices are scheduled on 10 processors, a performance gain of ≈ 1.46 s could be achieved, compared to the sequential LPT-scheduler (see Fig. 3b). Accordingly, the initial scheduling costs have been amortized after 24 SCF iterations.

6 Conclusion and Future Work

We demonstrated how a parallel eigensolver can be used efficiently in quantum chemistry software when the Hamilton matrix has a block-diagonal structure. The scalability of the proposed parallel scheduler has been demonstrated on real chemical systems. The first system, $Au_{55}(PH_3)_{12}$, scales as far as 20 processors. For the second system, Pd_{344}, a performance gain could be achieved until up to 28 processors. Compared to the previously used LPT-scheduler, the scalability was significantly improved. Performance improvements could be achieved by the factors of about 2 and 1.6, respectively, when using the same numbers of processors. With the improved parallelizability, the diagonalization step can now be executed about 8.4 and 4 times faster, respectively. Furthermore, the proposed strategy for the cost function, which relies on empiric records of execution times, provides results accurate enough for the scheduler to work in practice.

In summary, the technique presented significantly improves the performance and the scalability of the solution of the generalized eigenvalue problem in parallel Quantum Chemistry codes. It makes thus an important contribution to prevent this step from becoming a bottleneck in simulations of large symmetric molecular systems, especially nano particles.

It will be interesting to explore how an approximate scheduling algorithm like the one of [20] compares with the combinatorial approach proposed here. Adopting such an algorithm would also make the presented technique more versatile because the number of tasks would not be limited anymore. Thus, rare cases in practical applications with significantly more than 10 blocks would be covered as well.

Acknowledgments

The work was funded by the Munich Centre of Advanced Computing (MAC) and Fonds der Chemischen Industrie.

References

1. Altmann, S.L., Herzig, P.: Point-group theory tables. Clarendon, Oxford (1994)
2. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK's user's guide. SIAM, Philadelphia (1992)
3. Belling, T., Grauschopf, T., Krüger, S., Mayer, M., Nörtemann, F., Staufer, M., Zenger, C., Rösch, N.: High performance scientific and engineering computing. In: Bungartz, H.J., Durst, F., Zenger, C. (eds.) Lecture notes in Computational Science and Engineering, vol. 8, pp. 439–453 (1999)
4. Belling, T., Grauschopf, T., Krüger, S., Nörtemann, F., Staufer, M., Mayer, M., Nasluzov, V.A., Birkenheuer, U., Hu, A., Matveev, A.V., Shor, A.V., Fuchs-Rohr, M.S.K., Neyman, K.M., Ganyushin, D.I., Kerdcharoen, T., Woiterski, A., Majumder, S., Rösch, N.: ParaGauss, version 3.1. Tech. rep., Technische Universität München (2006)
5. Bischof, C., Müller, M.S., Knüpfer, A., Jurenz, M., Lieber, M., Brunst, H., Mix, H., Nagel, W.E.: Developing scalable applications with Vampir, VampirServer and VampirTrace. In: Proc. of ParCo 2007, pp. 113–120 (2007)
6. Blackford, L.S., Choi, J., Cleary, A., D'Azeuedo, E., Demmel, J., Dhillon, I., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C., Dongarra, J.: ScaLAPACK user's guide. SIAM, Philadelphia (1997)
7. Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., Weglarz, J.: Handbook on scheduling: from theory to applications. Springer, Heidelberg (2007)
8. Blazewicz, J., Kovalyov, M.Y., Machowiak, M., Trystram, D., Weglarz, J.: Scheduling malleable tasks on parallel processors to minimize the makespan. Ann. Oper. Res. 129, 65–80 (2004)
9. Decker, T., Lücking, T., Monien, B.: A 5/4-approximation algorithm for scheduling identical malleable tasks. Theor. Comput. Sci. 361(2), 226–240 (2006)
10. Demmel, J., Stanley, K.: The performance of finding eigenvalues and eigenvectors of dense symmetric matrices on distributed memory computers. In: Proc. seventh SIAM conf. on parallel processing for scientific computing, pp. 528–533 (1995)
11. Dunlap, B.I., Rösch, N.: The Gaussian-type orbitals density-functional approach to finite systems. Adv. Quantum Chem. 21, 317–339 (1990)

12. Garey, M., Johnson, D.: Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman and Company, New York (1979)
13. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. 45, 1563–1581 (1966)
14. Graham, R.: Bounds on multiprocessing timing anomalities. SIAM J. Appl. Math. 17, 263–269 (1969)
15. Hein, J.: Improved parallel performance of SIESTA for the HPCx Phase2 system. Tech. rep., The University of Edinburgh (2004)
16. Jansen, K.: Scheduling malleable parallel tasks: an asymptotic fully polynomial time approximation scheme. Algorithmica 39, 59–81 (2004)
17. Koch, W., Holthausen, M.C.: A chemist's guide to density functional theory. Wiley-VCH, Weinheim (2001)
18. Ludwig, W., Tiwari, P.: Scheduling malleable and nonmalleable parallel tasks. In: SODA 1994, pp. 167–176 (1994)
19. Mounié, G., Rapine, C., Trystram, D.: Efficient approximation algorithms for scheduling malleable tasks. In: SPAA 1999, pp. 23–32 (1999)
20. Mounié, G., Rapine, C., Trystram, D.: A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks. SIAM J. Comp. 37(2), 401–412 (2007)
21. National supercomputer HLRB-II, `http://www.lrz-muenchen.de/`
22. Steinberg, A.: A strip-packing algorithm with absolute performance bound 2. SIAM J. Comp. 26(2), 401–409 (1997)
23. Turek, J., Wolf, J., Yu, P.: Approximate algorithms for scheduling parallelizable tasks. In: SPAA 1992, pp. 323–332 (1992)
24. Ward, R.C., Bai, Y., Pratt, J.: Performance of parallel eigensolvers on electronic structure calculations II. Tech. rep., The University of Tennessee (2006)
25. Yudanov, I.V., Matveev, A.V., Neyman, K.M., Rösch, N.: How the C-O bond breaks during methanol decomposition on nanocrystallites of palladium catalysts. J. Am. Chem. Soc. 130, 9342–9352 (2008)
26. Yudanov, I.V., Metzner, M., Genest, A., Rösch, N.: Size-dependence of adsorption properties of metal nanoparticles: a density functional study on Pd nanoclusters. J. Phys. Chem. C 112, 20269–20275 (2008)