JCTC Journal of Chemical Theory and Computation

Article

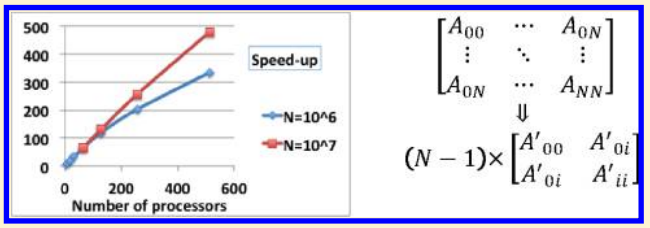
pubs.acs.org/JCTC

Diagonalization of Large Matrices: A New Parallel Algorithm

Ignacio Nebot-Gil*

Institute of Molecular Science, University of Valencia, c/Catedrático José Beltrán 2 E-46980-Paterna (Valencia), Spain

ABSTRACT: On the basis of a dressed matrices formalism, a new algorithm has been devised for obtaining the lowest eigenvalue and the corresponding eigenvector of large real symmetric matrices. Given an $N \times N$ matrix, the proposed algorithm consists in the diagonalization of $(N-1)2 \times 2$ dressed matrices. Both sequential and parallel versions of the proposed algorithm have been implemented. Tests have been performed on a Hilbert matrix, and the results show that this algorithm is up 340 times faster than the corresponding LAPACK routine for $N = 10^4$ and about 10% faster than the Davidson method. The parallel MPI version has been tested using up to 512 nodes. The speed-up for a $N = 10^6$ matrix is fairly lineal until 64 cores. The time necessary to obtain the lowest eigenvalue and eigenvector is nearly 5.5 min with 512 cores. For an $N = 10^7$ matrix, the speed-up is nearly linear to 256 cores and the calculation time is 5.2 h with 512 nodes. Finally, in order to test the new algorithm on MRCI matrices, we have calculated the ground state and the $\pi \rightarrow \pi^*$ excited state of the butadiene molecule, starting from both SCF and CASSCF wave functions. In all the cases considered, correlation energies and wave functions are the same as obtained with the Davidson algorithm.



1. INTRODUCTION

Matrix diagonalization is a very general problem in linear algebra, and the procedure used to solve a large scale eigenvalue equation is present in many physical, engineering, chemical, and applied mathematics numerical problems. More specifically, diagonalization of very large matrices is a major problem in quantum physics and quantum chemistry.

In quantum chemistry in particular, the full configuration interaction (FCI) procedure gives the exact solution for a given basis set.[1] FCI involves diagonalization of matrices whose dimensions increase nearly as $\binom{2N}{n}$ where N is the dimension of the basis set and n the number of electrons.[1] FCI calculations could easily attain billions of configurations. Therefore, FCI cannot be applied in most of the molecular systems and several strategies are used to truncate the FCI space. However, even truncated CI matrices, specially in MRCI calculations, can have very large dimensions, involving millions of determinants.

Several algorithms have been proposed to diagonalize large matrices, the Davidson method[2] being the most popular of molecular quantum chemistry programs. The Davidson algorithm is a variant of the more general Lanczos method.[3,4] Davidson and other similar algorithms are based on the successive multiplication of the matrix to be diagonalized by an approximate eigenvector. A new approximate eigenvector is then built by minimization of the residual vector, the difference between the approximate eigenvector and the exact one. Most quantum chemistry CI programs use different implementations of the Davidson procedure, in particular for solving FCI equations.[5−24]

In this paper, a new algorithm is proposed to obtain the lowest eigenvalue and the associate eigenvector of a square, symmetric, and real matrix, based on the dressed matrices formalism developed by Malrieu et al.[25] In section 2 the necessary theory is shown. The proposed algorithm is described in section 3. Section 4 includes the computational details. Finally, sections 5 and 6 describe the different implementation of the proposed algorithm and the results obtained with each of them, respectively.

2. THEORY: DRESSED MATRICES FORMALISM[25,26]

Let $\mathbf{A}$ be a square, real, and symmetric $N \times N$ matrix. Then, the eigenvalue equation for $\mathbf{A}$ can be written as

$$\mathbf{A}\mathbf{c}_k = \alpha_k \mathbf{c}_k \tag{1}$$

where and $\mathbf{c}_k$ and α_k are the kth eigenvector and eigenvalue, respectively. Assume that we are only interested in the lowest eigenvalue, α_0, and its corresponding eigenvector, $\mathbf{c}_0$, that is chosen to be in its intermediate normalization form. Then

$$\left[\begin{array}{cc|ccc} A_{00} & A_{01} & A_{02} & \cdots & A_{0N-1} \\ A_{01} & A_{11} & A_{12} & \cdots & A_{1N-1} \\ \hline A_{02} & A_{12} & A_{22} & \cdots & A_{2N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{0N-1} & A_{1N-1} & A_{2N-1} & \cdots & A_{N-1N-1} \end{array}\right] \left[\begin{array}{c} 1 \\ c_1 \\ \hline c_2 \\ \vdots \\ c_{N-1} \end{array}\right]_0 = \alpha_0 \left[\begin{array}{c} 1 \\ c_1 \\ \hline c_2 \\ \vdots \\ c_{N-1} \end{array}\right]_0 \tag{2}$$

According to the dressed matrix formulation of the effective Hamiltonian,[27] we can define a 2×2 model matrix $\mathbf{A}'$ from the upper right corner of $\mathbf{A}$:

Received: July 31, 2014
Published: December 22, 2014

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

$$\begin{bmatrix} A'_{00} & A'_{01} \\ A'_{01} & A'_{11} \end{bmatrix} \begin{bmatrix} 1 \\ c_1 \end{bmatrix} = \alpha_0 \begin{bmatrix} 1 \\ c_1 \end{bmatrix} \tag{3}$$

where the eigenvector of the model matrix is the projection of $\mathbf{c}_0$ on the model space, and the eigenvalue, α_0, is the same as in the complete matrix case. In eq 3, the subindex 0, indicating the eigenvector corresponding to lowest eigenvalue, has been dropped out, for simplicity.

The dressed matrices formalism[25,26] allows to define the model matrix $\mathbf{A}'$ as the sum of the upper right corner of $\mathbf{A}$ and a 2×2 dressing matrix, $\mathbf{\Delta}$:

$$\begin{bmatrix} A'_{00} & A'_{01} \\ A'_{01} & A'_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{01} & A_{11} \end{bmatrix} + \begin{bmatrix} \Delta_{00} & \Delta_{01} \\ \Delta_{01} & \Delta_{11} \end{bmatrix} \tag{4}$$

Therefore, the elements of the $\mathbf{\Delta}$ matrix can be obtained[26] by developing eq 3 and the two first rows of eq 2:

$$A'_{00} + A'_{01}c_1 = A_{00} + \Delta_{00} + A_{01}c_1 + \Delta_{01}c_1 = \alpha_0 \tag{5}$$

$$A'_{01} + A'_{11}c_1 = A_{01} + \Delta_{01} + A_{11}c_1 + \Delta_{11}c_1 = \alpha_0 c_1 \tag{6}$$

$$A_{00} + A_{01}c_1 + A_{02}c_2 + \cdots = A_{00} + A_{01}c_1 + \sum_{j>1}^{N-1} A_{0j}c_j = \alpha_0 \tag{7}$$

$$A_{01} + A_{11}c_1 + A_{12}c_2 + \cdots = A_{01} + A_{11}c_1 + \sum_{j>1}^{N-1} A_{1j}c_j = \alpha_0 c_1 \tag{8}$$

By comparing eq 5 with eq 7 and 6 with 8, respectively, one obtains

$$\sum_{j>1}^{N-1} A_{0j}c_j = \Delta_{00} + \Delta_{01}c_1$$

$$\sum_{j>1}^{N-1} A_{1j}c_j = \Delta_{01} + \Delta_{11}c_1 \tag{9}$$

Since in eq 3 we have only two unknowns, c_1 and α_0, we only need to define 2 of the 3 elements of $\mathbf{\Delta}$. Then, we have chosen to take $\Delta_{11} = 0$, and then the dressing matrix $\mathbf{\Delta}$ will have a matrix-column form.[28,29] This form keeps the symmetrical character for the dressing matrix, and avoids numerical troubles, since a diagonal form for the dressing matrix should involve the division by c_1.[28,29] Thus, the remaining elements of the dressing matrix in eq 9 will be

$$\Delta_{01} = \sum_{j>1}^{N-1} A_{1j}c_j$$

$$\Delta_{00} = \sum_{j>1}^{N-1} A_{0j}c_j - \Delta_{01}c_1 \tag{10}$$

The calculation of the $\mathbf{\Delta}$ elements involves, then, the knowledge of the whole eigenvector, $\mathbf{c}_0$. Therefore, an iterative procedure is needed to estimate the eigenvector elements, until convergence is attained.

In each iteration step, as it is well-known,[1] the lowest eigenvalue of the $\mathbf{A}$ matrix can be obtained as

$$\alpha_0 = \sum_{i=0}^{N-1} A_{0i}c_i = A_{00} + \sum_{i=1}^{N-1} A_{0i}c_i \tag{11}$$

3. PROPOSED ALGORITHM

The eigenvalue of the $\mathbf{A}$ matrix can be obtained from eq 11, which assumes that all the $N - 1$ elements of the eigenvector $\mathbf{c}_0$ are known. The proposed algorithm obtains each of these elements, say c_i, by solving the corresponding eigenvalue equation for the dressed 2×2 matrices of the form:

$$\begin{bmatrix} A'_{00} & A'_{0i} \\ A'_{0i} & A_{ii} \end{bmatrix} \begin{bmatrix} 1 \\ c_i \end{bmatrix} = \alpha_0 \begin{bmatrix} 1 \\ c_i \end{bmatrix} \quad i = 1, N-1 \tag{12}$$

Since the full vector $\mathbf{c}_0$ is necessary to calculate the dressing matrix elements, an iterative procedure is used.

In the zero-th iteration, we assume that $\mathbf{\Delta} = \mathbf{0}$. Then, we obtain a first-order approximation to the eigenvector, $\mathbf{c}_0^{(0)}$, by solving the following $(N-1)2 \times 2$ eigenvalue equations:

$$\begin{bmatrix} A_{00} & A_{0i} \\ A_{0i} & A_{ii} \end{bmatrix} \begin{bmatrix} 1 \\ c_i^{(0)} \end{bmatrix} = \alpha_0 \begin{bmatrix} 1 \\ c_i^{(0)} \end{bmatrix} \quad i = 1, N-1 \tag{13}$$

Then, for a given iteration I, $(I \neq 0)$, the $(N-1)$ eigenvalue equations of the 2×2 dressed matrices, $\mathbf{A}'$, to be solved, are of the following form:

$$\begin{bmatrix} A'^{(I)}_{00} & A'^{(I)}_{0i} \\ A'^{(I)}_{0i} & A_{ii} \end{bmatrix} \begin{bmatrix} 1 \\ c_i^{(I+1)} \end{bmatrix} = \alpha_0^{(I+1)} \begin{bmatrix} 1 \\ c_i^{(I+1)} \end{bmatrix} \quad i = 1, N-1 \tag{14}$$

Where the corresponding dressing matrix elements, $\Delta_{0i}^{(I)}$ and $\Delta_{00}^{(I)}$, are calculated from the previous iteration eigenvector, $\mathbf{c}_0^{(I)}$

$$\Delta_{0i}^{(I)} = \sum_{j\neq i,0}^{N-1} A_{ij}c_j^{(I)}$$

$$\Delta_{00}^{(I)} = \sum_{j\neq 0,i}^{N-1} A_{0j}c_j^{(I)} - \Delta_{0i}^{(I)}c_i^{(I)} \tag{15}$$

A simpler expression for evaluating $\Delta_{00}^{(I)}$ can be obtained after comparing eqs 10 and 11. So, the dressed matrix element $A'^{(I)}_{00}$ can be calculated as

$$A'^{(I)}_{00} = A_{00} + \Delta_{00}^{(I)} = A_{00} + \sum_{j\neq 0,i}^{N-1} A_{0j}c_j^{(I)} - \Delta_{0i}^{(I)}c_i^{(I)} \tag{16}$$

after adding and subtracting $A_{0_i}c_i^{(I)}$, one obtains

$$\begin{aligned} A'^{(I)}_{00} &= [A_{00} + A_{0i}c_i^{(I)} + \sum_{j\neq 0,i}^{N-1} A_{0j}c_j^{(I)}] - \Delta_{0i}^{(I)}c_i^{(I)} - A_{0i}c_i^{(I)} \\ &= \alpha_0^{(I)} - \Delta_{0i}^{(I)}c_i^{(I)} - A_{0i}c_i^{(I)} \end{aligned} \tag{17}$$

hence, the A'_{00} element of the dressed matrix can be calculated from the eigenvalue obtained in the previous iteration.

Solving eq 14 involves a simple second-order equation, as described below.

After all the $(N-1)c_j^{(I+1)}$ coefficients have been calculated, the eigenvalue, $\alpha_0^{(I+1)}$, can be also obtained as

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

$$\alpha_0^{(I+1)} = A_{00} + \sum_{i=1}^{N-1} A_{0i} c_i^{(I+1)} \tag{18}$$

Then, if $|\alpha_0^{(I+1)} - \alpha_0^{(I)}|$ is smaller than the desired accuracy, the process ends. Else, a new iteration is performed.

Two improvements have been devised in order to accelerate the convergence:

1. Since the coefficient vector, $\mathbf{c}_0$, is taken in its intermediate normalization form, we can update it when each c_i is calculated. We have found that this procedure can decrease the number of iterations in one.
2. Let the matrix elements of A ordered in such a way that $c_1 \geq \cdots \geq c_{i-1} \geq c_i \geq c_{i+1} \geq \cdots \geq c_{N-1}$. We have found that, at the Ith iteration, the best order for calculating them is the smallest coefficient, $c_{N-1}^{(I)}$, is calculated first and the greater one, $c_1^{(I)}$, is last. Thus, the dressing matrix element $\Delta_{0N-1}^{(I-1)}$ necessary to obtain $c_{N-1}^{(I)}$ is calculated with the coefficients of the previous iteration $\mathbf{c}_0^{(I-1)}$. However, the coefficient $c_i^{(I)}$ is obtained using a mix of the coefficients $\{1, c_1^{(I-1)}, \ldots, c_i^{(I-1)}, c_{i+1}^{(I)}, \ldots, c_{N-1}^{(I)}\}$ calculated in the previous iteration and in the same iteration. In this way, the most important coefficient, $c_1^{(I)}$, will include the effects of the coefficients obtained in the same iteration, $\{1, c_1^{(I-1)}, c_2^{(I)}, \ldots, c_{N-1}^{(I)}, c_N^{(I)}\}$. This procedure can also decrease the required number of iterations in one.

In order to obtain the coefficients, each eigenvalue equation in eq 12 is expanded as a set of two equations:

$$A'_{00} + A'_{0i} c_i = \alpha_0 \tag{19}$$

$$A'_{0i} + A_{ii} c_i = \alpha_0 c_i \tag{20}$$

After eliminating α_0, one obtains a second-order equation of the following form:

$$c_i^2 + \left(\frac{A'_{00} - A_{ii}}{A'_{0i}}\right) c_i - 1 = 0 \tag{21}$$

And, taking $K = (A'_{00} - A_{ii})/A'_{0i}$, the solutions of the above equation are[30]

$$c_{i_1} = q$$

$$c_{i_2} = \frac{-1}{q} \tag{22}$$

where

$$q = -\frac{1}{2}[K + \mathrm{sign}(K)\sqrt{K^2 + 4}] \tag{23}$$

Then, we choose the solution c_{i_1} or c_{i_2} having the smallest absolute value, i.e., the one having the absolute value smaller than 1, since the intermediate normalization of $\mathbf{c}_0$ involves that all the $|c_i| \leq 1$.

Then, two asymptotic limits must be considered:

1. Limit of $|c_i| \to 0$: If $A'_{0i} \ll (A'_{00} - A_{ii})$ in eq 21, K is large enough as compared with 4, and therefore, $q = -K$. We will take $c_i = (1/K) = 0$, to avoid numerical unstabilities.
2. Limit of $|c_i| \to 1$: If $A'_{00} \approx A_{ii}$, then $K \to 0$ and $|c_i| \to 1$. In this case, the eigenvector is not dominated by only one coefficient. Then, it should be necessary to increase the model space to a $M \times M$ matrix, where $M - 1$ is the number of coefficients near the unity. The diagonal of the model matrix should include all the corresponding dressed matrix elements.

We could also calculate c_i by means of a even simpler form, by isolating it from eq 20 and using the eigenvalue from the previous iteration, as follows:

$$c_i^{(I+1)} = \frac{A_{0i}^{\prime(I)}}{\alpha_0^{(I)} - A_{ii}} = \frac{A_{0i} + \Delta_{0i}^{(I)}}{\alpha_0^{(I)} - A_{ii}} \tag{24}$$

One must be aware from the possible numerical unstabilities arising from the condition, $\alpha_0 \approx A_{ii}$.

4. COMPUTATIONAL DETAILS

The proposed algorithm has been tested on a version of the Hilbert matrix,[31] where each matrix element is calculated from its corresponding indexes:

$$A_{ii} = -\frac{1}{2 \quad i+1} \quad i = 0, N-1$$

$$A_{ij} = -\frac{1}{\gamma(i+j+1)}$$

$$i = 0, N-1 \quad j = 0, N-1 \quad i \neq j \tag{25}$$

where $\gamma = 10$.[31]

In our test calculations we have used the following computers:

- AMM: Apple MacMini, with an Intel Core 2 Duo processor at 2.4 GHz, with 4 GB of RAM memory and a NVIDIA GeForce 320 M GPU with 256 MB of RAM.
- VIVES: A SGI Altix UltraViolet 1000 with 64 CPU Xeon series 7500 hexacore (384 cores in total, but only 48 are accessible at the same time), at 2.67 GHz, with 960 GB of RAM (Computer Center of the University of Valencia).
- TIRANT: An IBM computer with 512 blades JS21 with 2 processors PowerPC 970MP dual core at 2.2 GHz and 4 GB of RAM each, interconnected by means of a Myrinet network (this is the University of Valencia node of the Spanish Supercomputation Network, RES).

5. IMPLEMENTED ALGORITHMS

The general algorithm proposed in this work, as described above, is sketched in Figure 1. The proposed algorithm has been implemented in three versions, which differ in the memory usage.

5.1. Explicitly Calculated Matrix. In this case the matrix elements are calculated and stored as a double precision $N \times N$ array. Then, the total storage needed is $8N(N+1)$ bytes, $8N$ for the double precision coefficient vector and the rest for the matrix elements.

The most time-consuming section of the algorithm is the inner loop over $j = 1, N-1$, that represents a matrix-vector product $\mathbf{Ac} = \mathbf{\Delta}$ to obtain the $\mathbf{\Delta}$ vector. To optimize this part, we have put this section in an outer loop and calculate $\mathbf{\Delta}$ by means of the efficient DGEMV BLAS routine[32−36] for multiplying a matrix and a vector (see Figure 2).

In this case the storage needed is $8N(N+2)$ bytes, where the additional $8N$ bytes are needed to store the $\mathbf{\Delta}$ dressing vector elements.

In order to have a comparison of the efficiency of the implemented algorithms we have also programmed the diagonalization of the same Hilbert matrix as above by means of the LAPACK DSYEVX and DSPEVX diagonalization routines,[37]

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

```
alpha=0.d0
alpha0=A(0,0)
iter=0
do while abs(alpha-alpha0).gt.threshold
   alpha0=alpha
   Ap00=A(0,0)
   Ap0i=A(0,i)
   Apii=A(i,i)
   do i=N-1,1,-1 ! Inverse order
      if (iter.ne.0) then
         Delta0i=0.d0
         do j=1,N-1
            if (j.ne.i) Delta0i=Delta0i+A(i,j)*c(j)
         end do
         Ap00=alpha0-A(0,i)*c(i)-Delta0i*c(i)
         Ap0i=Ap0i+Delta0i
      end if
      ...
      ! resolution of the 2x2 dressed matrix
      ! to obtain c(i)
      ...
      alpha=alpha+c(i)*A(0,i)
   end do
   iter=iter+1
end do
```

Figure 1. Algorithm scheme.

```
alpha=0.d0
alpha0=A(0,0)
iter=0
do while abs(alpha-alpha0).gt.threshold
   alpha0=alpha
   if (iter.ne.0) then
      call DGEMV(Delta,A,coef)
      alpha=Delta(0)
   end if
   Ap00=A(0,0)
   Ap0i=A(0,i)
   Apii=A(i,i)
   do i=1,N-1
      if (iter.ne.0) then
         Delta(i)=Delta(i)-A(0,i)-A(i,i)*c(i)
         Ap00=alpha-A(0,i)*c(i)-Delta(i)*c(i)
         Ap0i=Ap0i+Delta(i)
      end if
      ...
      ! resolution of the 2x2 dressed matrix
      ! to obtain c(i)
      ...
   end do
   iter=iter+1
end do
```

Figure 2. Algorithm scheme using BLAS DGEMV routine.[32−36]

for obtaining only the lowest eigenvector and corresponding eigenvalue of a square symmetric matrix.

5.2. Calculation of the Matrix Upper Triangle. In order to reduce the storage needed, only the matrix upper triangle is stored as a vector. In this case, only $8[N(N+1)/2+N]$ bytes are needed, $8N$ for the coefficient vector in double precision and the rest for the matrix upper triangle.

In order to have a comparison of the efficiency of the implemented algorithms we have also programmed the diagonalization of the Hilbert matrix by means of the LAPACK DSPEVX diagonalization routine,[37] for obtaining only the lowest eigenvector and corresponding eigenvalue of a square symmetric matrix, when only the matrix upper triangle is stored in the form of a vector.

5.3. Direct Algorithm. *5.3.1. Sequential Implementation.* Finally, and in order to treat very large matrices, we have implemented the algorithm in a direct version. Thus, each matrix element is calculated when needed, by means of an appropriate double precision function. Therefore, only $8N$ bytes of storage are needed to store the coefficient vector elements in double precision.

5.3.2. Parallel Implementations. Since the bottleneck in the algorithm is the calculation of the dressing matrix elements inside a double loop of dimensions $(N-1)^2$, it seems advisible to parallelize the inner loop. Two approaches have been chosen. One uses the Open MP (multiprocessing) application programming interface (API)[38] and the other the MPI (message passing interface) API.[39]

In the case of the Open MP Parallel version, we have included the corresponding directives to transform the inner loop in a parallel loop, as shown in Figure 3.

```
!$ parallel shared (...) private (...) reduction (+:...)
!$ do
      do j=1,N-1
         if (j.ne.i) Delta0i=Delta0i+A(i,j)*c(j)
      end do
!$ omp end do
!$ omp end parallel
```

Figure 3. Open MP implementation of the algorithm.

For the MPI parallel version, we need to initialize the MPI environment and then to obtain the rank of each processor and the number of processors involved (N_{procs}).

Then, at the beginning of each iteration, the master processor broadcasts (by means of a MPI_BROADCAST subroutine call) the coefficient vector to all the slave processors. Then, the inner loop is split over all the processors in such a way that each runs over $(N-1)/N_{procs}$ elements to calculate the contribution of the corresponding dressing matrix element. The master processor collects all the contributions, by means of a call to MPI_REDUCE subroutine, and then calculates the coefficient corresponding to the outer loop index.

6. RESULTS AND DISCUSSION

6.1. Sequential Algorithms. The results for different matrix sizes are collected on Table 1 and Figure 4. A MiniMac computer (AMM, as described in section 4) has been used. The programs in this section have been compiled with the version 4.6.2 of the gFortran compiler[40] with $-O3$ optimization level. The threshold for convergence was chosen to be 10^{-6} for all cases, and typically the required number of iterations is 4−5.

Table 1 collects the "user" time, as given by the *time* UNIX command, for each program and matrix size. The different columns corresponds to the following algorithm versions:

- DH-S-V: the algorithm version which stores the matrix upper triangle as a vector as described in section 5.2.
- DH-S-D: the direct algorithm version, which calculates each matrix element when needed as described in section 5.3.1.
- DH-S-M: the algorithm version which calculates all the matrix and uses double precision BLAS routines to calculate the dressing vector elements as described in section 5.1.
- DLAP-M: the Hilbert matrix is calculated, as in the DH-S-M version, and then diagonalized for only the

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

Table 1. Elapsed Times (s) of Diagonalization of Hilbert Matrices ($N \times N$) with the Proposed Sequential Algorithm (DH-S) and Comparison with LAPACK Diagonalization Rutines

N	DH-S-V[a]	DH-S-D[b]	DH-S-M[c]	DLAP-M[d]	DLAP-V[e]	Dav-D[f]
1×10^1	0.001	0.002	0.001	0.001	0.001	
1×10^2	0.002	0.002	0.001	0.003	0.002	
1×10^3	0.063	0.036	0.028	0.918	0.828	0.040
2×10^3	0.280	0.137	0.109	7.696	8.240	0.157
3×10^3	0.657	0.308	0.234	25.805	28.651	0.346
4×10^3	1.207	0.543	0.384	60.809	69.103	0.619
5×10^3	1.929	0.851	0.601	118.295	136.668	0.966
6×10^3	2.833	1.228	0.871	203.648	236.572	1.386
7×10^3	3.915	1.673	1.268	322.722	369.675	1.887
8×10^3	5.220	2.177	1.691	480.689	549.761	2.463
9×10^3	6.702	2.758	2.136	683.393	837.962	3.128
1×10^4	8.464	3.425	2.691	927.229	1176.324	3.848
1×10^5		339.770				368.694

[a]Matrix upper triangle is calculated and stored as a vector. [b]Matrix elements are calculated as needed. [c]Whole matrix is calculated and stored. Dressing vector is calculated with the BLAS routine. [d]LAPACK diagonalization routine. Whole matrix is calculated and stored. [e]LAPACK diagonalization routine. Matrix upper triangle is calculated and stored as a vector. [f]Davidson diagonalization routine. Matrix elements are calculated as needed.

lowest eigenvector, by means of the corresponding double precision LAPACK routine.

- DLAP-V: as in the previous case, but only the matrix upper triangle is calculated and stored as a vector.
- Dav-D: the diagonalization is performed by means of the Davidson algorithm.[2] Each matrix element is calculated when needed.

For implementations storing the matrix, the results show that, in all cases, the dressing matrix algorithms are faster than those using LAPACK routines and, moreover, the calculation time grows slowly. Due to the limited RAM of the computer (4 GB) only matrices up to size $10^4 \times 10^4$ can be calculated. The DH-S-M algorithm is the fastest one, taking only 2.691 s for a $10^4 \times 10^4$ matrix. Comparison of the algorithms that calculate the whole matrix (DH-S-M and DLAP-M) shows a time relation of more than 340 in favor of the dressing algorithm for the largest matrix. For the algorithms calculating only the matrix upper triangle (DH-S-V and DLAP-V), the time relation is nearly 140 in favor of the DH algorithm.

Comparison of the direct algorithms (DH-S-D and Dav-D) shows similar values for the computation times, although the DH algorithm is nearly a 10% faster. Both algorithms behave clearly as $O(N^2)$ ones. In this case, we have also calculated the lowest eigenvalue and eigenvector for a $10^5 \times 10^5$ matrix, in only 339.77 s.

The computation times for the three versions of the dressing algorithm and the Davidson method are presented in Figure 5. Comparison of the values shows that the DH-S-M version is the fastest, although the direct version is only 1.3 times slower for the largest matrix.

Table 2 shows for each matrix size, the absolute values of the differences between the lowest eigenvalue calculated with each program and the DLAP-M ones, which are also given as reference. The average of the differences for each program is also given. In all cases, the absolute value of the differences is smaller than the threshold.

6.2. Parallel Algorithms. *6.2.1. Open MP Calculations: DH-P-D-OMP.* The calculation times, obtained by means of *omp_get_wtime()*, the OMP elapsed time routine, are shown on Table 3. Calculations have been performed for matrices of dimensions 10^4, 10^5, and 10^6 using the Open MP version of the direct dressing matrix algorithm on the VIVES computer (see section 4). Since each processor has six cores and due to the SGI NUMA parallel architecture, the best results are obtained when using an even number of processors. Therefore, we have

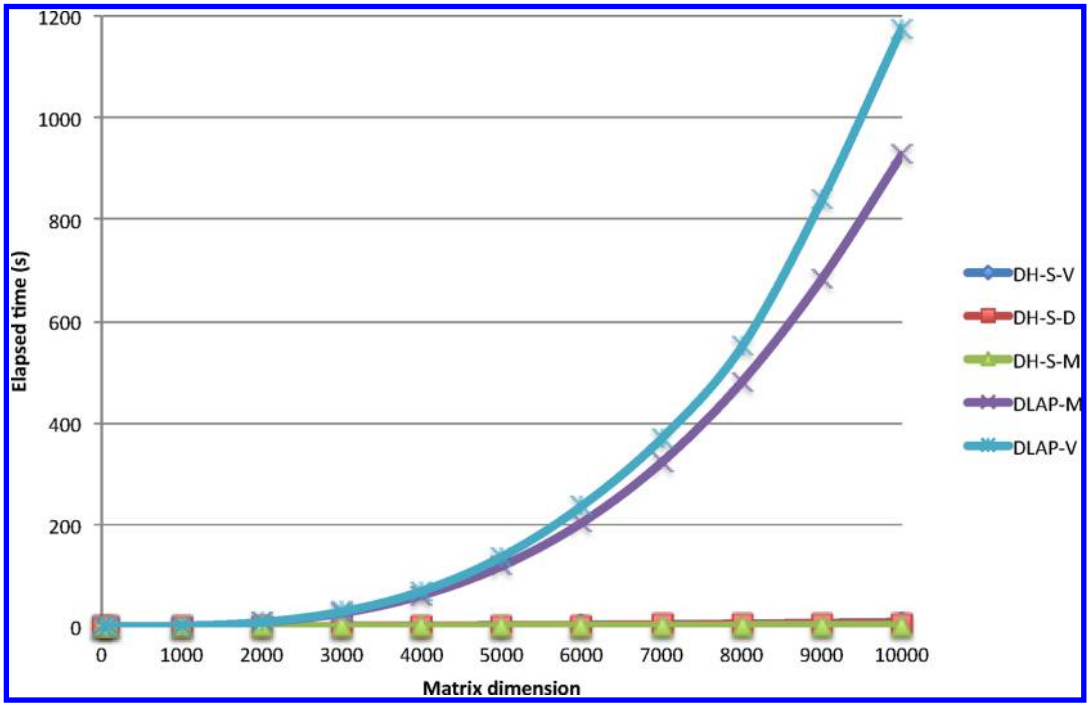


Figure 4. Elapsed times (s) of diagonalization of Hilbert matrices ($N \times N$) with the proposed sequential algorithm (DH-S) and comparison with LAPACK diagonalization routines (DLAP).

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

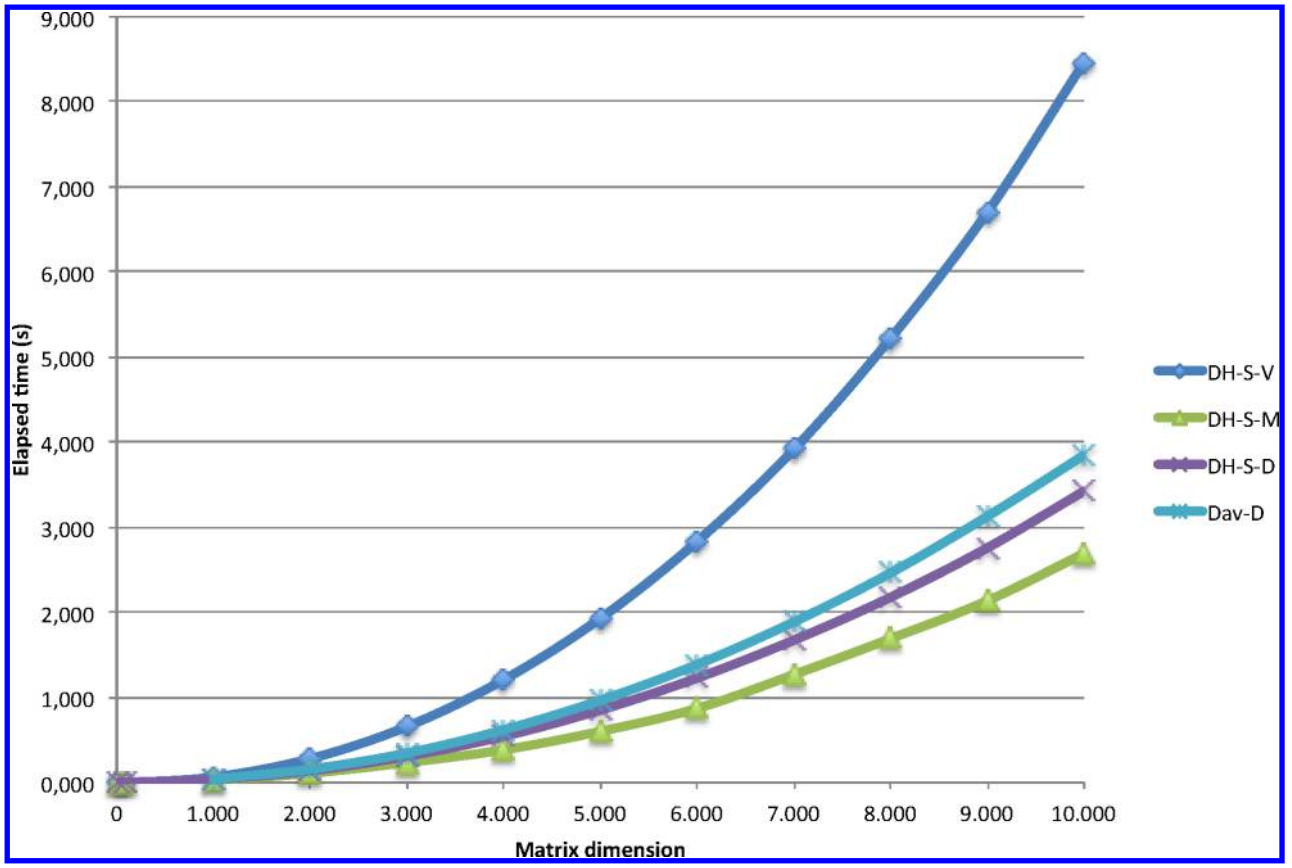


Figure 5. Elapsed times (s) of diagonalization of Hilbert matrices ($N \times N$) with three versions of proposed sequential algorithm (DH-S) and comparison with Davidson diagonalization algorithm (Dav).

Table 2. Absolute Value of the Differences between the Eigenvalue Obtained with the Different Algorithms and Corresponding to DLAP-M (Convergence Threshold 10^{-6})

N	DH-S-V[a]	DH-S-D[b]	DH-S-M[c]	DLAP-V[d]	Dav-D[e]	DLAP-M[f]
1×10^1	2.86×10^{-7}	2.86×10^{-7}	3.46×10^{-7}	7.04×10^{-7}		−1.00789701
1×10^2	3.35×10^{-7}	3.35×10^{-7}	4.90×10^{-7}	5.27×10^{-7}		−1.00933613
1×10^3	7.64×10^{-8}	7.64×10^{-8}	2.91×10^{-8}	1.01×10^{-7}	8.58×10^{-8}	−1.00956710
2×10^3	2.89×10^{-7}	2.89×10^{-7}	1.61×10^{-7}	3.51×10^{-7}	3.02×10^{-7}	−1.00958549
3×10^3	8.26×10^{-8}	8.26×10^{-8}	2.22×10^{-7}	1.52×10^{-7}	6.80×10^{-8}	−1.00959282
4×10^3	1.59×10^{-9}	1.59×10^{-9}	1.49×10^{-7}	1.50×10^{-7}	1.43×10^{-8}	−1.00959647
5×10^3	2.77×10^{-7}	2.77×10^{-7}	1.24×10^{-7}	4.48×10^{-7}	2.94×10^{-7}	−1.00959856
6×10^3	2.02×10^{-7}	2.02×10^{-7}	4.47×10^{-8}	1.94×10^{-7}	2.19×10^{-7}	−1.00960027
7×10^3	2.99×10^{-7}	2.99×10^{-7}	1.39×10^{-7}	1.72×10^{-7}	3.17×10^{-7}	−1.00960138
8×10^3	1.32×10^{-7}	1.32×10^{-7}	3.16×10^{-8}	6.69×10^{-8}	1.50×10^{-7}	−1.00960249
9×10^3	2.68×10^{-7}	2.68×10^{-7}	4.34×10^{-7}	1.23×10^{-7}	2.49×10^{-7}	−1.00960363
1×10^4	1.28×10^{-8}	1.28×10^{-8}	1.56×10^{-7}	2.15×10^{-7}	3.17×10^{-8}	−1.00960396
average	1.88×10^{-7}	1.88×10^{-7}	1.94×10^{-7}	2.67×10^{-7}	1.73×10^{-7}	

[a]Matrix upper triangle is calculated and stored as a vector. [b]Matrix elements are calculated as needed. [c]Whole matrix is calculated and stored. Dressing vector is calculated with the BLAS routine. [d]LAPACK diagonalization routine. Matrix upper triangle is calculated and stored as a vector. [e]Davidson diagonalization routine. Matrix elements are calculated as needed. [f]DLAP-M eigenvalue. LAPACK diagonalization routine. The whole matrix is calculated and stored.

chosen a number of cores of 6, 12, 24, 36, and 48 (1, 2, 4, 6, and 8 processors). The values for one core are given in order to calculate the speedup and the efficiency of the algorithm for each matrix dimension. Speedup for N cores is defined as the relation between the time necessary for a calculation using one core and the time necessary for the calculation using N cores:[41]

$$S(N) = \frac{T_1}{T_N} \tag{26}$$

Efficiency is defined as speedup divided by the number of cores:[41]

$$E(N) = \frac{S(N)}{N} \tag{27}$$

In general, the speedup of a parallel algorithm grows initially in a nearly lineal form with the number of cores. As the number of cores grows, the linearity is lost, and, finally, the speedup reaches the Amdahl's law limit.[42] The larger the Amdahl's law

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

Table 3. Open-MP Parallel Version of the Direct Algorithm (DH-P-D-OMP): Elapsed time in Seconds for Obtaining the Lowest Eigenvector and Eigenvalue of Matrices of Dimension $N \times N$ (Convergence Threshold 10^{-6}, VIVES Shared Memory Computer)

N_{procs}	$N = 10^4$	$N = 10^5$	$N = 10^6$
1	4.346	432.269	43429.121
6	1.058	73.583	7219.876
12	1.154	41.749	3659.001
24	6.424	38.585	2018.836
36	5.328	54.332	1592.717
48	10.524	131.767	1392.735

limit, the better the parallel algorithm. Efficiency is considered to be good enough if its value is greater than 0.5.

The resulting speedup and efficiency values are shown in Figures 6 and 7. The results show that for smaller matrices the parallelization involves an improvement only for a reduced number of cores. For instance, for a matrix of dimensions of $10^5 \times 10^5$, increasing the number of cores beyond 12 cores does not reduces the calculation time. Speedup is good for 12 cores but decreases for a larger number of cores. For an actually large matrix $(10^6 \times 10^6)$ the speedup and efficiency behavior is good. It seems to indicate than the Amdahl's law limit will be reached beyond 80 cores. The eigenvalues obtained are independent of the number of cores and very close to the reference value for each matrix dimension.

6.2.2. MPI Calculations: DH-P-D-MPI. Calculations have been performed for matrices of dimensions 10^5, 10^6, and 10^7 using the MPI version of the direct dressing matrix algorithm in the VIVES memory-shared computer (see section 4). As above, we have chosen a number of cores of 6, 12, 24, 36, and 48 (1, 2, 4, 6, and 8 processors). The calculation times, obtained from the MPI elapsed time routine, *MPI_WTIME()*, are shown on Table 4. The values for 1 core are given in order to calculate the speedup and the efficiency of the algorithm for each matrix dimension, as defined in the previous subsection.

Speedup and efficiency of the parallel algorithm are shown in Figures 8 and 9. The results show that the parallelization involves a true improvement when one increases the number of cores. Speedup for the $(10^5 \times 10^5)$ matrix shows that the Amdahl's limit law could be reached beyond 80 cores. For a larger matrix $(10^6 \times 10^6)$ speedup is fairly linear with the number of cores, in such a way that, for 48 cores, the time necessary to obtain the lowest eigenvalue and eigenvector is 1/47 the time necessary for 1 core. It is to point out that the time necessary to obtain the lowest eigenvalue and eigenvector is nearly 15.5 min. We have also tried to calculate the lowest eigenvalue and eigenvector for a $(10^7 \times 10^7)$ matrix, with 48 cores. The time necessary has been 25.15 h.

This version of the algorithm opens the door for obtaining eigenvalues of very large matrices with a true parallel program. Therefore, in order to try the behavior of the algorithm in very large matrices and many processors, calculations have been performed for matrices of dimensions $10^6 \times 10^6$ and $10^7 \times 10^7$ using the MPI version of the dressing matrix direct algorithm in the TIRANT distributed memory computer (see section 4) until 512 processors. The calculation times, obtained from the MPI elapsed time routine, *MPI_WTIME()*, are shown in Table 5. Since each processor has 2 cores, we have chosen a number of cores of 2^n, $n = 0, 3, 4, ..., 9$ for the $10^6 \times 10^6$ matrix and $n = 6, ..., 9$ for the $10^7 \times 10^7$ matrix. The value for one core is given in order to calculate the speedup and the efficiency of the algorithm as defined in a previous subsection.

When the number of parallel processes increases, the communication among them can stall the calculation. In order to avoid stalling at the MPI_REDUCE subroutine call, it is convenient to synchronize all the parallel processes by introducing a call to the subroutine MPI_BARRIER. Each processor, when arrives to the barrier, stops and waits until all processes have

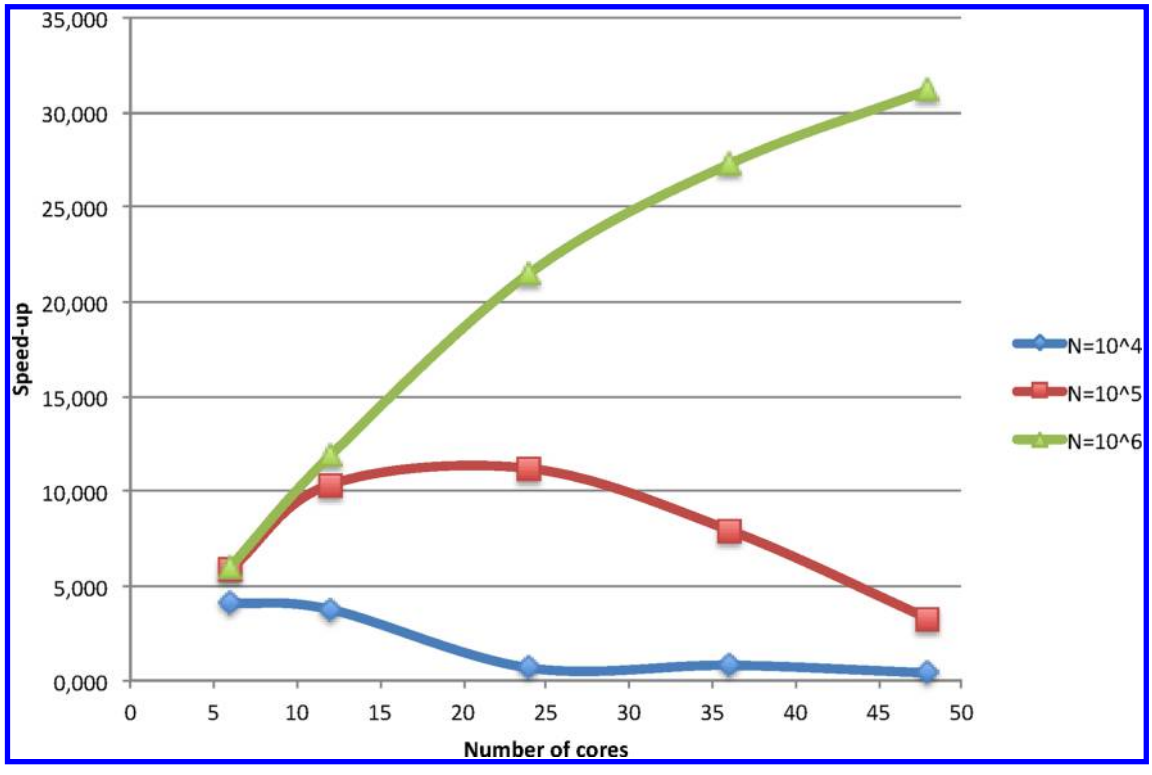


Figure 6. Speedup of the DH-P-D-OMP algorithm applied to matrices of dimensions $(N \times N)$.

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

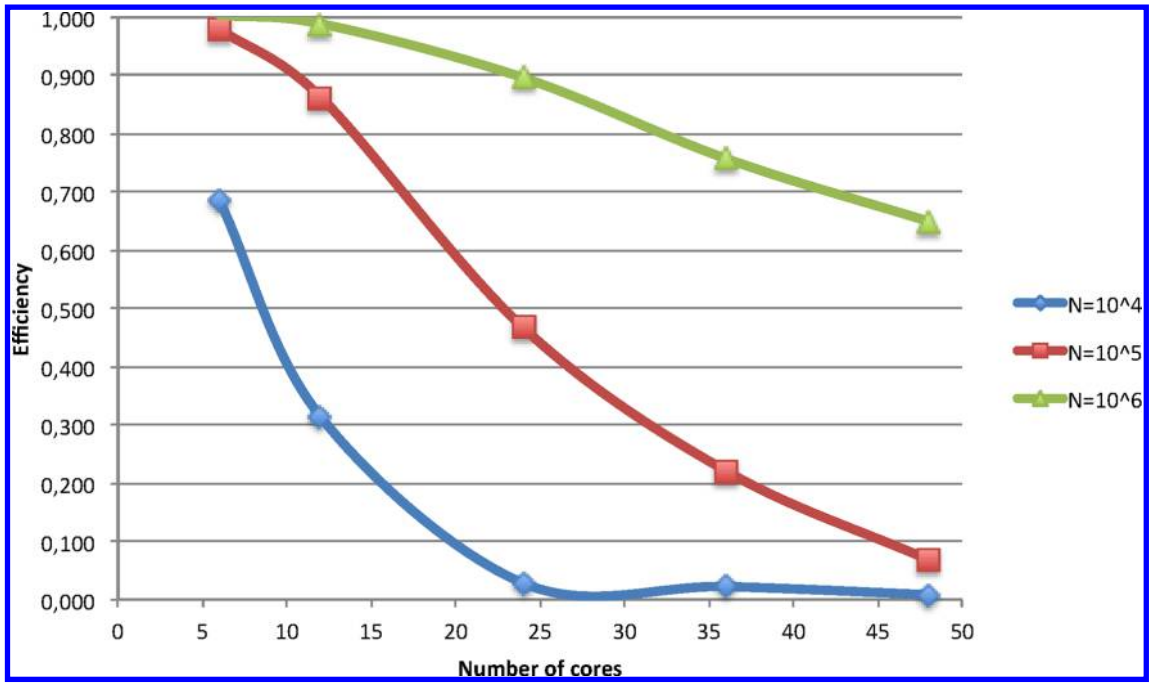


Figure 7. Efficiency of the DH-P-D-OMP algorithm applied to matrices of dimensions ($N \times N$).

Table 4. MPI Parallel Version of the Direct Algorithm (DH-P-D-MPI): Elapsed Time in Seconds for Obtaining the Lowest Eigenvector and Eigenvalue of Matrices of Dimension $N \times N$ (Convergence Threshold 10^{-6}, VIVES Shared Memory Computer)

N_{procs}	$N = 10^5$	$N = 10^6$	$N = 10^7$
1	435.859	43486.318	
6	75.374	7211.986	
12	40.648	3629.472	
24	23.158	1832.707	
36	16.996	1228.642	
48	14.076	925.933	90640.938

Table 5. MPI Parallel Version of the Direct Algorithm (DH-P-D-MPI): Elapsed Time in Seconds for Obtaining the Lowest Eigenvector and Eigenvalue of Matrices of Dimension $N \times N$ on the TIRANT Distributed Memory Computer (Convergence Threshold 10^{-6})

N_{procs}	$N = 10^6$	$N = 10^7$
1	111440.629	8929343.715[a]
8	13780.784	
16	6853.969	
32	3462.906	
64	1771.074	139520.996
128	939.575	68802.141
256	552.190	34949.473
512	333.879	18673.194

[a]Estimated from the 64 processors value.

arrived to same point. Then, all processes restart. Thus, the inner loop is calculated as described in Figure 10.

The synchronization, by means of an MPI_BARRIER call, is performed for each value of *i*. This trick introduces a complete synchronization among all parallel processes.

However, if the synchronization is performed in all the inner N-loop steps, the average waiting time becomes very high. It has been empirically shown that if the synchronization is performed each, say, 25 steps, the overall performance is far better, as summarized in Figure 11.

The results show that the parallelization involves a true improvement when one increases the number of cores, although the TIRANT PowerPC cores are nearly 2.5 times slower than the VIVES Xeon ones. Speedup and efficiency of the algorithm in this computer are shown in Figures 12 and 13. Speed-up for the $10^6 \times 10^6$ matrix is fairly linear with the number of cores up to 64 cores, where the time necessary to obtain the lowest eigenvalue and eigenvector is 1/63 the time necessary for one core. It is to point out that the time necessary to obtain the lowest eigenvalue and eigenvector is nearly 5.5 min with 512 cores and the calculation converges in only 4 iterations, due to the use of the most efficient algorithm for calculation the coefficients in reverse order. Moreover, the more efficient synchronization procedure involves an improvement going from a 16% for 64 cores to a 44% for 512 cores. The speed-up is nearly linear until 256 cores, and the asymptotic limit of the Amdahl law would be attained beyond 1300 cores.

6.3. CI Test Calculations. In order to try the reliability of the proposed algorithm for diagonalizing CI matrices, test MRCI calculations have been performed on the butadiene molecule. Two different starting wave functions have been chosen to build the CI matrices: the SCF wave function and a CASSCF one built taking four active electrons in the following four active π orbitals: $1a_u$, $1b_g$, $2a_u$, and $2b_g$. The SCF and CASSCF calculations have been performed with the MOLCAS 7.4 program[44] and using the ANO-L basis sets for C and H atoms.[45] Then, CAS+S CI matrices have been explicitly built by means of the CASDI series of programs[43] from both the SCF and CASSCF wave functions. The CAS+S CI space includes all the CAS determinants, as defined previously, plus all the possible single excited determinants built from the CAS ones, having a dimension of 12 748 determinants. The matrices so obtained have then been diagonalized by means of the Davidson algorithm and of the DH-S-M algorithm proposed in this paper (see sections 5.1 and 6.1).

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

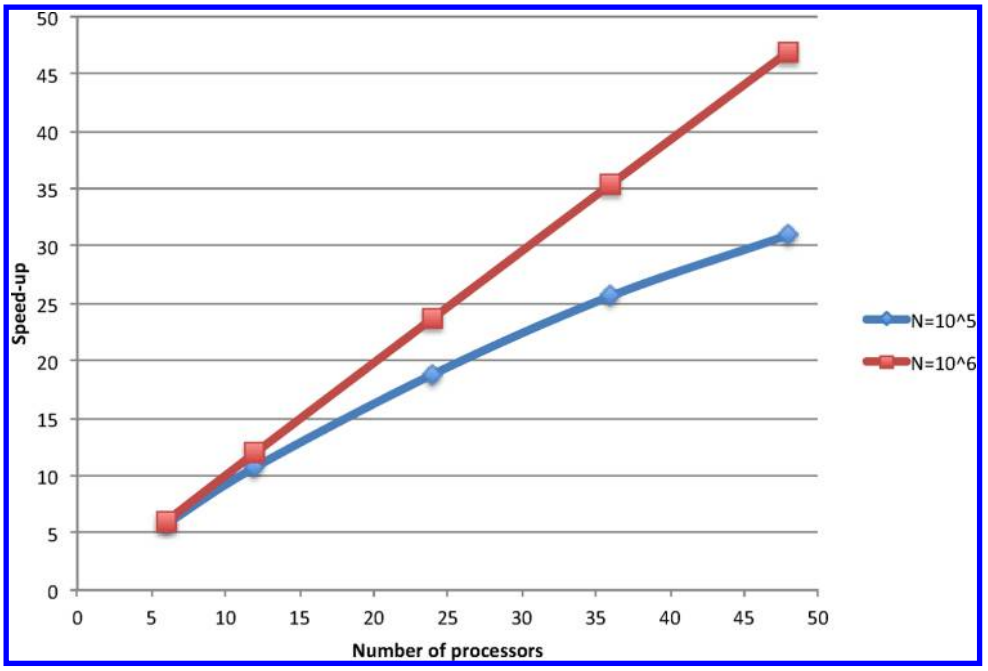


Figure 8. Speed-up of the DH-P-D-MPI algorithm applied to matrices of dimensions ($N \times N$) on the VIVES shared memory computer.

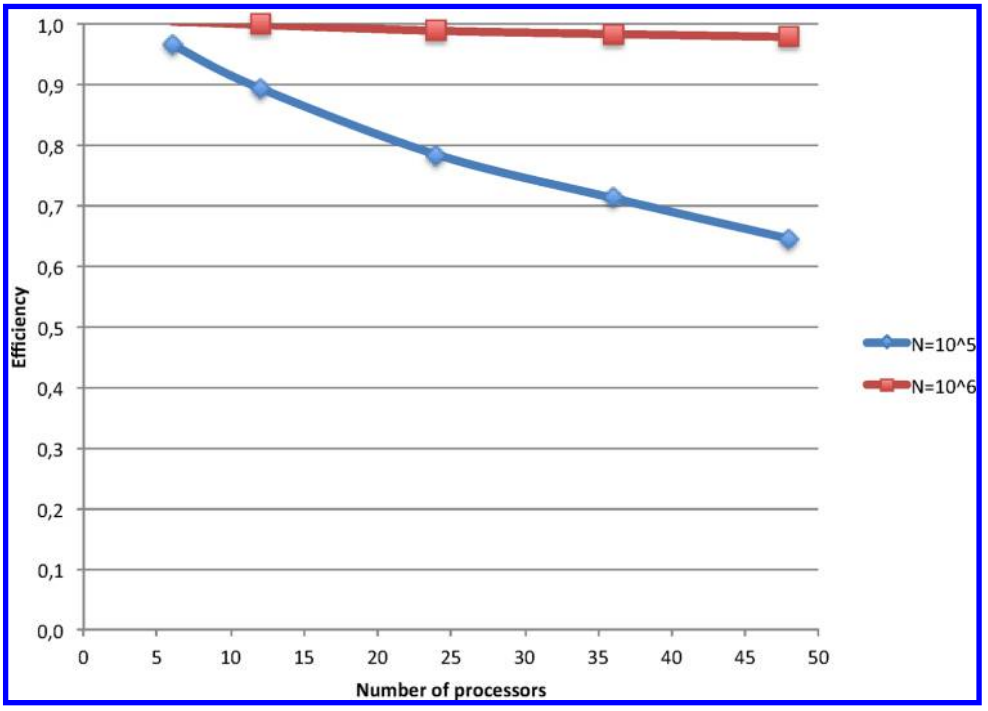


Figure 9. Efficiency of the DH-P-D-MPI algorithm applied to matrices of dimensions ($N \times N$) on the VIVES shared memory computer.

```
...
do i=1,N-1
   ...
   call MPI_BARRIER(...)
   call MPI_REDUCE(...)
   ...
end do
...
```

Figure 10. Synchronization scheme.

Both the ground state and the lowest $\pi \rightarrow \pi^*$ excited state have been calculated. The results are summarized in Table 6. In the ground state case, the closed-shell determinant, $|\Psi_0\rangle$, has been chosen as reference to build all the 2×2 matrices in eq 12. In the case of the excited state, we have chosen as reference the main single determinant corresponding to the single excitation, as follows:

- $\overline{1b_g} \rightarrow \overline{2b_g}$ for the SCF starting wave function.
- $\overline{1a_u} \rightarrow \overline{2a_u}$ for the CASSCF starting wave function.

In all the cases, the proposed algorithm yields the same results as the Davidson algorithm, the differences being of the order of 10^{-8} for the eigenvalues and 10^{-5} for the eigenvectors' main coefficients (see Table 6). It is to point out that for the $\pi \rightarrow \pi^*$ excited state, the proposed algorithm is able to deal

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

```
...
icount=0
do i=1,N-1
   ...
   icount=icount+1
   if (icount.eq.25) then
      icount=0
      call MPI_BARRIER(...)
   end if
   call MPI_REDUCE(...)
   ...
end do
...
```

Figure 11. Improved synchronization scheme.

with a multireference wave function, since, despite selecting only one determinant as reference, it obtains the right result: four determinants having similar coefficients that correspond to the two main configurations: HOMO − 1 → LUMO ($1a_u \rightarrow 2a_u$ and $\overline{1a_u} \rightarrow \overline{2a_u}$) and HOMO → LUMO + 1 ($1b_g \rightarrow 2b_g$ and $\overline{1b_g} \rightarrow \overline{2b_g}$).

6.4. Concluding Remarks. A very simple algorithm for obtaining the lowest eigenvector and its corresponding eigenvalue of large real and symmetric matrices, based on the dressing matrices formalism,[25,26] has been devised. The algorithm consists in the diagonalization of $(N - 1)2 \times 2$ dressed matrices. The bottleneck of the proposed algorithm is the obtention of the elements of the dressing matrix, involving a N^2 loop. Comparison with standard LAPACK diagonalization routines shows that the proposed algorithm is by far faster. Also, it is slightly faster when compared with the Davidson method.

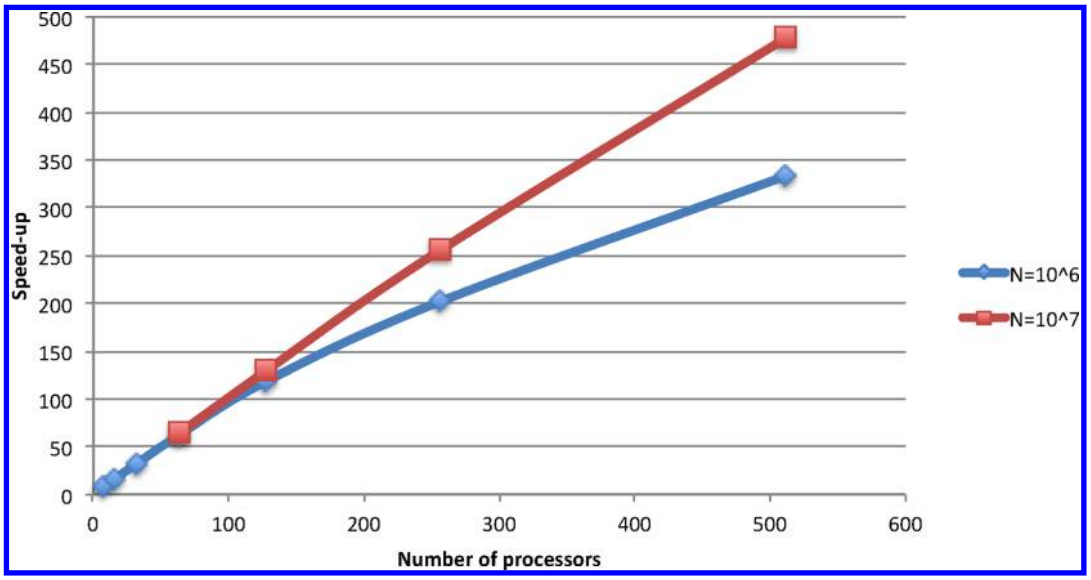


Figure 12. Speed-up of the DH-P-D-MPI algorithm applied to matrices of dimensions $(N \times N)$ on the TIRANT distributed memory computer.

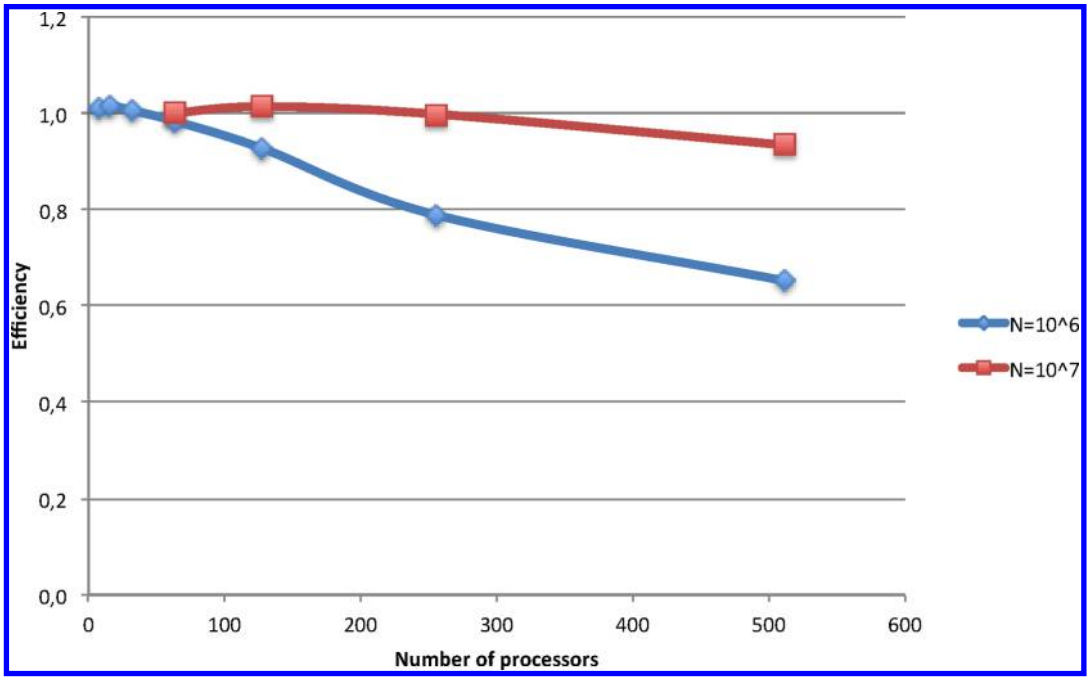


Figure 13. Efficiency of the DH-P-D-MPI algorithm applied to matrices of dimensions $(N \times N)$ on the TIRANT distributed memory computer.

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

Table 6. CAS+S CI Calculations on the Two Lowest Electronic States of Butadiene[a]

starting wave function	SCF[b]		CASSCF (4e in 4OM)[c]	
		Ground State		
eigenvalue (a.u.)	−0.09952601[d]	−0.09952600[e]	−0.13711665[e]	−0.13711664[e]
eigenvector[f]				
$\|\Psi_0\rangle$	0.9490064[d]	0.9490162[e]	0.9442299[d]	0.9442409[e]
$(1b_g)^2 \rightarrow (2a_u)^2$	−0.1582411[d]	−0.1581987[e]	−0.1617004[d]	−0.1616399[e]
		$\pi \rightarrow \pi^*$ Excited State		
eigenvalue (a.u.)	0.08801198[d]	0.08801200[e]	0.06695283[d]	0.06695284[e]
eigenvector[f]				
$\overline{1b_g} \rightarrow \overline{2b_g}$	0.4833118[d]	0.4833201[e]	0.4538142[d]	0.4537818[e]
$1b_g \rightarrow 2b_g$	−0.4833115[d]	−0.4833151[e]	−0.4538144[d]	−0.4538064[e]
$\overline{1a_u} \rightarrow \overline{2a_u}$	0.4539217[d]	0.4539079[e]	0.4882213[d]	0.4882472[e]
$1a_u \rightarrow 2a_u$	−0.4539219[d]	−0.4539122[e]	−0.4882210[d]	−0.4882466[e]
$\overline{1b_g} \rightarrow \overline{3b_g}$	0.1124528[d]	0.1124659[e]		
$1b_g \rightarrow 3b_g$	−0.1124528[d]	−0.1124647[e]		

[a]Comparison of the proposed algorithm with the Davidson algorithm. [b]E(SCF) = −154.87607910 au. [c]E(CASSCF) = −154.93689611 au. [d]Proposed algorithm (DH-S-M, section 6.1). [e]Davidson algorithm. [f]Only the coefficients with values $|c_i| \geq 0.1$ are shown.

The simplicity of the method allows easy parallelization, since that the calculation of the dressing matrix elements can be distributed over different calculation nodes. To do that MPI strategy has proven to be more efficient than the OMP one. Calculations with $N = 10^6$ and $N = 10^7$ and up 512 nodes show a fair linearity of the speedup versus the number of nodes.

The test has been carried out on a version of the dense Hilbert matrix. However, all the strategies devised to work on sparse matrices with a known structure, as done for instance in the FCI programs,[9,14,16,18,20] can be also easily introduced in this algorithm. In this way, this algorithm can be adapted to most of existing CI programs. In fact, at each iteration of the diagonalization process, the needed dressing matrix elements form a vector, $\boldsymbol{\Delta}$

$$\Delta_{0i}^{(I)} = \sum_{j \neq i,0}^{N-1} A_{ij} c_j^{(I)} \tag{28}$$

which is obtained by the multiplication of the $\mathbf{A}$ matrix by the coefficient's vector, $\mathbf{c}$. This step is the same time-consuming as in the Davidson algorithm. Many Davidson algorithm implementations substitute the $O(N^2)$ step ($\mathbf{Ac}$) by a procedure taking benefit of the a priori well-known block structure of the CI matrix and, then, only the nonzero blocks are explicitly taken into account, for instance, by substituting the long loop over the determinants by a shorter loop over the integral lists, where only the contribution of each integral to the corresponding elements of Δ is calculated.[22,43]

Therefore, the algorithm proposed in this paper can adopt the same strategies to accelerate the dressing matrix elements calculation and the diagonalization procedure shown here can then substitute the usual Davidson procedure[2] or, at least, be used to obtain good trial vectors for the Davidson procedure.

Then, a direct implementation of the proposed algorithm can be devised. Only four vectors are needed to be kept in memory to store the diagonal elements of the matrix, the matrix row corresponding to the state searched, the coefficient vector, and the dressing vector. The implementation could be as follows:

1. Calculation of the matrix diagonal $\{A_{ii}\}$ and the row associated with the target eigenvector $\{A_{0i}\}$.
2. Initialize the iteration counter $n = 0$.
3. Calculation of the zero-th order coefficient vector, $\mathbf{c}^{(0)}$, by solving the $(N-1)2 \times 2$ undressed matrices.
4. Actualize the iteration counter $n = n + 1$.
5. Loop over the integral lists to calculate directly the contribution of each integral to the elements of the dressing vector $\boldsymbol{\Delta} = \mathbf{Ac}^{(n)}$, for the nth iteration.
6. Calculation of the nth coefficient vector, $\mathbf{c}^{(n)}$, by solving the $(N-1)2 \times 2$ dressed matrices.
7. Verification of the convergence. If the process converges, it stops; else, it performs another iteration.

A clear additional advantage of the proposed algorithm, due to its simplicity, is the smaller operations number involved, giving as a consequence smaller rounding errors and better numerical stability.

The tests carried out on the butadiene molecule indicate that the algorithm is reliable to treat MRCI matrices built from either monoconfigurational or CASSCF wave functions and for obtaining monoreference as well as multireference states. It seems reasonable to expect, therefore, that the algorithm could be even applied also to dressed CI matrices.[26] Also, it is possible to implement a state-independent dressing[46] to obtain simultaneously n ($n < N$) states, which should involve the diagonalization of $N - n$ dressed $(n + 1) \times (n + 1)$ matrices.

AUTHOR INFORMATION

Corresponding Author

*E-mail: Ignacio.Nebot@uv.es.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

The author wishes to thank J. P. Malrieu for many fruitful discussions and D. Maynau for kindly providing us the CASDI program. The author acknowledges also the financial support of the Ministerio de Ciencia e Innovación (ref. CTQ2010-19738). The computer time in parallel computers was provided by the Red Española de Supercomputación (RES) and the Computer Center of the University of Valencia.

DEDICATION

Devoted to Prof. Jean-Paul Malrieu on his 75th birthday.

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483

REFERENCES

(1) Szabo, A.; Ostlund, N. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*; Dover Books on Chemistry Series; Dover Publications, 1996; p 232.
(2) Davidson, E. R. *J. Comput. Phys.* **1975**, *17*, 87−94.
(3) Lanczos, C. *J. Res. Natl. Bur. Stand. (U. S.)* **1950**, *45*, 255−282.
(4) Lanczos, C. *J. Res. Natl. Bur. Stand. (U. S.)* **1952**, *49*, 33−53.
(5) Handy, N. C. *Chem. Phys. Lett.* **1980**, *74*, 280−283.
(6) Saxe, P.; Shaefer, H. F., III; Handy, N. C. *Chem. Phys. Lett.* **1981**, *79*, 202−204.
(7) Harrison, R. J.; Handy, N. C. *Chem. Phys. Lett.* **1983**, *95*, 386−391.
(8) Harrison, R.; Handy, N. *Chem. Phys. Lett.* **1983**, *98*, 97−101.
(9) Siegbahn, P. E. M. *Chem. Phys. Lett.* **1984**, *109*, 417−423.
(10) Knowles, P. J.; Handy, N. C. *Chem. Phys. Lett.* **1984**, *111*, 315−321.
(11) Bauschlicher, C. W.; Taylor, P. R. *J. Chem. Phys.* **1986**, *85*, 2779−2783.
(12) Bauschlicher, C. W.; Langhoff, S. R.; Partridge, H.; Taylor, P. R. *J. Chem. Phys.* **1986**, *85*, 3407−3410.
(13) Graham, R. L.; Yeager, D. L.; Olsen, J.; Jørgensen, P.; Harrison, R.; Zarrabian, S.; Bartlett, R. *J. Chem. Phys.* **1986**, *85*, 6544−6549.
(14) Olsen, J.; Roos, B. O.; Jørgensen, P.; Jensen, H. J. A. *J. Chem. Phys.* **1988**, *89*, 2185−2192.
(15) Bauschlicher, J.; Charles, W.; Langhoff, S. R. *Theor. Chim. Acta* **1988**, *73*, 43−53.
(16) Knowles, P. J.; Handy, N. C. *J. Chem. Phys.* **1989**, *91*, 2396−2398.
(17) Zarrabian, S.; Sarma, C.; Paldus, J. *Chem. Phys. Lett.* **1989**, *155*, 183−188.
(18) Knowles, P. J. *Chem. Phys. Lett.* **1989**, *155*, 513−517.
(19) Harrison, R. J.; Zarrabian, S. *Chem. Phys. Lett.* **1989**, *158*, 393−398.
(20) Olsen, J.; Jørgensen, P.; Simons, J. *Chem. Phys. Lett.* **1990**, *169*, 463−472.
(21) Bendazzoli, G.; Evangelisti, S. *Chem. Phys. Lett.* **1991**, *185*, 125−130.
(22) Bendazzoli, G. L.; Evangelisti, S. *J. Chem. Phys.* **1993**, *98*, 3141−3150.
(23) Olsen, J.; Jørgensen, P.; Koch, H.; Balkova, A.; Bartlett, R. J. *J. Chem. Phys.* **1996**, *104*, 8007−8015.
(24) Bauschlicher, C. W.; Langhoff, S. R.; Taylor, P. R. *Adv. Chem. Phys.*; John Wiley & Sons, Inc., 2007; pp 103−161.
(25) Daudey, J.-P.; Heully, J.-L.; Malrieu, J.-P. *J. Chem. Phys.* **1993**, *99*, 1240−1254.
(26) Sánchez-Marn, J.; Nebot-Gil, I.; Malrieu, J.-P.; Heully, J.-L.; Maynau, D. *Theor. Chim. Acta* **1996**, *95*, 215−241.
(27) Malrieu, J.-P.; Durand, P.; Daudey, J.-P. *J. Phys. A: Math. Gen.* **1985**, *18*, 809−826.
(28) Nebot-Gil, I.; Sánchez-Marín, J.; Malrieu, J.-P.; Heully, J.-L.; Maynau, D. *J. Chem. Phys.* **1995**, *103*, 2576−2588.
(29) Nebot-Gil, I.; Sánchez-Marín, J.; Heully, J.-L.; Malrieu, J.-P.; Maynau, D. *Chem. Phys. Lett.* **1995**, *234*, 45−49.
(30) Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. *Numerical Recipes 3rd ed.: The Art of Scientific Computing*, 3rd ed.; Cambridge University Press: New York, NY, USA, 2007; p 178.
(31) Anglada, J.-M.; Besalú, E.; Bofill, J.-M. *Theor. Chem. Acc.* **1999**, *103*, 163−166.
(32) Lawson, C. L.; Hanson, R. J.; Kincaid, D. R.; Krogh, F. T. *ACM Trans. Math. Software* **1979**, *5*, 308−323.
(33) Dongarra, J. J.; Croz, J. D.; Hammarling, S.; Hanson, R. J. *ACM Trans. Math. Software* **1988**, *14*, 1−17.
(34) Dongarra, J. J.; Croz, J. D.; Hammarling, S.; Hanson, R. J. *ACM Trans. Math. Software* **1988**, *14*, 18−32.
(35) Dongarra, J. J.; Croz, J. D.; Hammarling, S.; Duff, I. *ACM Trans. Math. Software* **1990**, *16*, 1−17.
(36) Dongarra, J. J.; Croz, J. D.; Hammarling, S.; Duff, I. *ACM Trans. Math. Software* **1990**, *16*, 18−28.
(37) http://www.netlib.org/lapack. LAPACK is a software package provided by Univ. of Tennessee; Univ. of California, Univ. of Colorado Denver, and NAG Ltd., Berkeley (accessed September 17, 2014).
(38) http://openmp.org/wp/ (accessed September 17, 2014).
(39) Gabriel, E.; Fagg, G. E.; Bosilca, G.; Angskun, T.; Dongarra, J. J.; Squyres, J. M.; Sahay, V.; Kambadur, P.; Barrett, B.; Lumsdaine, A.; Castain, R. H.; Daniel, D. J.; Graham, R. L.; Woodall, T. S. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. *Proceedings 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, Sept 19−22, 2004; pp 97−104.
(40) https://gcc.gnu.org/fortran/ (accessed September 17,2014).
(41) Bentz, J. Parallel Computing. *WolframMathWorld*. Weisstein, E. W.; http://mathworld.wolfram.com/ParallelComputing.html (accessed September 17, 2014).
(42) Amdahl, G. M. Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings AFIPS*. **1967**, 483−485.
(43) CASDI program: Ben Amor, N.; Maynau, D. *Chem. Phys. Lett.* **1998**, *286*, 211−220.
(44) Molcas 7.4: Aquilante, F.; De Vico, L.; Ferré, N.; Ghigo, G.; Malmqvist, P.-Å.; Neogrády, P.; Pedersen, T. B.; Pitonak, M.; Reiher, M.; Roos, B. O.; Serrano-Andrés, L.; Urban, M.; Veryazov, V.; Lindh, R. *J. Comput. Chem.* **2010**, *31*, 224−247.
(45) Widmark, P.-O.; Malmqvist, P.-Å.; Roos, B. O. *Theor. Chim. Acta* **1990**, *77*, 291−306.
(46) Malrieu. J. P., private communication.

DOI: 10.1021/ct500689u
J. Chem. Theory Comput. 2015, 11, 472−483