**Open Access Library Journal, 2014, 1, 1-15**
Published Online May 2014 in OALib. http://www.oalib.com/journal
http://dx.doi.org/10.4236/oalib.1100411

OALib

# Parallel Calculation of the Electron Correlation Energy

**E. Ramos**

Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Valencia, Spain
Email: ramos@dsic.upv.es

Received 5 April 2014; revised 10 May 2014; accepted 21 May 2014



## Abstract

**Calculation of electron correlation energy in molecules involves a very important computational effort, even in the simplest cases. Nowadays, using the new parallel libraries (PETSc and SLEPc) and MPI, we can resolve this calculation faster and with very big molecules. This result is a very important advance in chemical computation.**

## Keywords

**Electron Correlation Energy, Chemical Molecular Properties Chemical Computation, Large-Scale Eigenvalue Problem, SLEPc, PETSc, Message-Passing Parallelization, MPI**

## 1. Introduction

Quantum Chemistry is a highly computational demanding discipline that has proved to be extremely useful in the qualitative and semiquantitative interpretation in chemistry. Calculations of Molecular Orbitals are almost a routine at present. They can even be found in many experimentalist articles on chemical reactivity.

However, nowadays, the challenge relies on chemical accuracy. Many efforts are devoted to calculate energy differences between different points of the same potential energy surfaces, or between different potential energy surfaces. These values represent relevant chemical magnitudes, such as reaction energies, activation energies or spectroscopic transitions. The challenge consists in calculating them with the same exactitude as the precision of the experimental values to be compared. Since experimental accuracy is nearly 1 kcal/mol, the calculations would be as accurate as 1 part per $10^5$ to $10^7$, depending on the size of the molecule.

The basis for almost all molecular calculations is the molecular orbitals (MO) calculated by means of the Hartree-Fock-Roothaan Self Consistent Field (SCF) procedure. However, the approximate nature of the mono-electron potential in the SCF procedure seriously limits the quantitative applicability of its results. Due to the

fact that correlated movements of the electrons in the molecule should be taken into account. This movement involves energy, the correlation energy, EC, which is 100 to 1000 times greater that the chemical precision.

The methods to calculate EC are based on three different approaches: Configuration Interaction (CI), the Many Body Perturbation Theory and the Coupled Cluster (CC) theories [1]. Other works have proposed several techniques to take into account the highest value of EC. Two of these methods, due to their noniterative nature, are specially suitable for parallelization. The first one is based on perturbative calculation over a SDCI wave function [2]. The second is based on the dressed matrices technique. With this method, the Hamiltonian matrix spanned by a model space is dressed or modified by an addition of the appropriate terms in the diagonal or in some rows or columns in order to incorporate the effects due the outer space (mainly, triply excited, T, and quadruply excited, Q determinants). Then, the average value of this dressed Hamiltonian is calculated. It has been shown that this is a very powerful technique to calculate EC [3]. We call this method Mean Value Total Dressing, MVTD.

Both methods can be programmed in a very similar way, and a large part of the resulting code, here called TQ2, is common.

Thanks to PETSc and SLEPc parallel libraries working with MPI, we have improved the performance a faster and optimal parallelization of these codes with a very important result. At this moment we can calculate the Electron Correlation Energy faster and in bigger molecular systems. This work presents these results.

## 2. Theoretical Background

The calculation of the Electron Correlation Energy in molecules is one of the fundamental problems of Computer Chemistry. The objective is to consider the simultaneous movement and connection of all the electrons in the electric field generated by the nucleus. The electron calculation is done for a position fixed or nailed to the nucleus (Born-Oppenheimer approximation) [4]. In the theoretical study of a chemical reaction, it is fundamental to evaluate the energetic contribution of the dynamic system formed by electrons with the purpose of obtaining precise values of the difference of energy between products and reagents. This takes place in the simplest case, where a new chemical bond is the reaction of two molecules or atoms, A and B:

$$\mathrm{A}+\mathrm{B}\rightarrow \mathrm{A}-\mathrm{B} \tag{1}$$

The energy present in the chemical process is only one small fraction (smaller than 0.01%) of the total energy of A and B. It is clear the necessity of very precise methods to evaluate the electron energy, since the accumulated error in the estimation of the energies of A, B and A − B, increases the error in the estimation of $E_{A-B}-\left(E_A+E_B\right)$.

From the point of view of the Quantum Mechanics, the correlation energy of a molecule (for example, A in the normal state) can be defined as the difference between the exact electron energy $E_A$ and the best estimation of the energy evaluated at monoelectron level, that is to say, with a method of calculation that only considers an equally divided way [1] of the collective effect of the other electrons in each of them. The energy of the monoelectron model, denominated *Energy Limit of Hartree-Fock*, $E^{HF}$ [4], is:

$$E_A^{corr}=E_A-E_A^{HF} \tag{2}$$

In **Figure 1** we can see the different energies:

- (a) Correlation Energy;
- (b) Complete CI Energy;
- (c) Experimental Energy.

These differences show the necessity to calculate with the highest possible exactitude the correlation energy.

## 3. TQ2 Program

TQ2 program has been implemented [2] [3] based on variations to obtain a physically acceptable estimation of the correlation energy. TQ2 requires a necessary chain of calculations to obtain the mentioned energy. The succession of calculations necessary to obtain the electron correlation energy is the following [4]:

**1) Orbital integrals calculation between atomic:** starting off the nature of the atoms that form the molecule and its geometry (its disposition in the space) and using a base of constructed functions as Gaussian linear combinations of $\varphi_j$ with the following form:

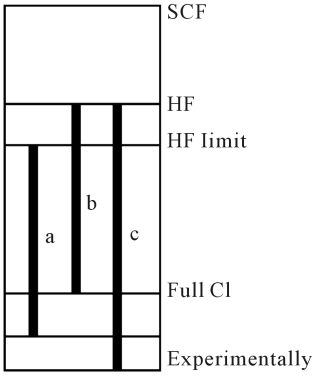


**Figure 1.** Energy comparation.

$$\varphi_j = \sum_j a_{kj} g_k \tag{3}$$

The calculations of integrals one, two and three are needed in the following stages.

**2) Orbital calculation of molecular ones:** they are obtained as a linear combination of Orbital Atomic:

$$\psi_i = \sum_j c_{kj} \varphi_k \tag{4}$$

The electrons are described by means of the space functions and spin:

$$\chi_i = \psi_i \sigma_i \tag{5}$$

In order to obtain the Orbital Molecular ones, the Fock's operator is calculated, which requires the integrals obtained in the previous stage.

**3) Coefficients $c_{kj}$:** they are obtained by iterative optimization of the reference energy, $E^{HF}$, formulated as:

$$E^{HF} = \langle \varphi_0 | H | \varphi_0 \rangle \tag{6}$$

where $H$ is the electron one Hamiltonian and $\varphi_0$ are the Fock's operator diagonalization function. This function $\varphi_0$ is calculated in the form of determinants of Salter:

$$\phi_0 = \left| \chi_1(1), \chi_2(2), \cdots, \chi_n(n) \right| \tag{7}$$

When the iterative process converges, Orbital Molecular and $E^{HF}$ of reference are obtained simultaneously.

**4) Calculation of the electron correlation energy:** HF is obtained in the Equation (6).

Great part of the calculation carried out in the three first steps has been solved by using the chain of HONDO programs [5], which is very effective for the mentioned calculations, but it solves them in a sequential way.

## Introduction to Total Dressing

TD is based on the model of effective Hamiltonian [6], where an N-dimensional reference model SDTQ is used. The wave function SDTQCI is:

$$\Psi = \phi_0 + \sum_{i \in [SD]} c_i \phi_i + \sum_{\alpha \notin SD} c_\alpha \phi_\alpha \tag{8}$$

where the space is being partitioning in:

**1) Space main model**, $\phi_0$, that is the wave function Hartree-Fock.

**2) Space intermediate model**, that includes all the mono and diexcitations obtained from $\phi_0$.

**3) External space**, constituted of tri and quadriexcitations of $\phi_0$.

An operator of suitable coating $\Delta$ is:

$$\varepsilon_0 = \langle \Psi | P(H + \Delta) P | \Psi \rangle \tag{9}$$

where $P$ is a projection operator that excludes the components from $\Psi$ in the external space:

$$P = \sum_{i \in S} |\phi_i\rangle\langle\phi_i| \tag{10}$$

With this formulation, it is possible to reduce the problem to the following standard eigenvalues equation with $N = \dim(S) - 1$ values [7]. For the resolution we only need to define $N$ elements of $\mathbf{\Delta}$, leaving the rest with value zero:

$$\left[\mathbf{P}(\mathbf{H} + \mathbf{\Delta})\mathbf{P} - \varepsilon_0 \mathbf{P1P}\right]\mathbf{c} = \mathbf{0} \tag{11}$$

Several definitions of $\mathbf{\Delta}$ are:

**1) Diagonal Dressing:**

$$\begin{gathered} \Delta_{ii} = \frac{1}{c_i} \sum_{\alpha \in S} c_\alpha H_{i\alpha} \\ \Delta_{ij} = 0 \end{gathered} \tag{12}$$

**2) Hermitic Row-Column Dressing:**

$$\begin{gathered} \Delta_{i0} = \Delta_{0i} = \sum_{\alpha \notin S} c_\alpha H_{i\alpha} \\ \Delta_{00} = \sum_{\alpha \notin S} c_\alpha H_{0\alpha} - \sum_{i \neq 0} \Delta_{i0} c_i \\ \Delta_{ij} = 0 \end{gathered} \tag{13}$$

In Equations (12) and (13), the values of $\mathbf{\Delta}$ come from the vector **c**, which it is obtained in Equation (11). It is therefore necessary to use an iterative procedure until reaching the auto-consistency. Several approximate procedures have been developed to evaluate the coefficients $c_\alpha$ from the set of coefficients $c_i$. For this it is necessary that the coefficients $c_i$ are sufficiently next to the exact values. The coefficients $c_i$ resulting from SDCI are not good because the wave function SDCI leads to an estimation of the Correlation Energy that does not grow linearly with the number of electrons. The resulting error is very important, being able to lead to the 100% of errors in systems of about 30 electrons. A wave function that it has corrected this defect can be obtained in the coating scheme and is known s the function $(SC)^2$ −SDCI . It is obtained calculating vector **c** of the Equation (11) using the following coating diagonal:

$$\Delta_{ii} = \sum_{j \in D_i} c_j H_{0j} \tag{14}$$

where $D_i$ is the set of all the diexcitations that can be done on $\phi_i$ and $\phi_0$. The *dressing* or coating (*SC*) is a partial coating, because it only includes in the estimation of $c_\alpha$ the effects of the essential correlation to obtain the correct dependancy of the energy with the number of electrons.

Available *Total Dressing* methods are [7]:

**1) Perturbative Evaluation of the Coefficients $c_\alpha$:** The coefficients are calculated according to equation:

$$c_\alpha = \frac{1}{D_\alpha} \sum_{i \in S} c_i \langle \phi_i | H | \phi_\alpha \rangle \tag{15}$$

where $D_\alpha$ is a denominator of Epstein-Nesbet type (Method TD1),

$$D_\alpha = H_{00} - H_{\alpha\alpha} \tag{16}$$

An additional correction allows us to apply a displacement $EPV_\alpha$ (method TD1EPV):

$$D_\alpha = H_{00} - H_{\alpha\alpha} - EPV_\alpha \tag{17}$$

**2) *Coupled Cluster* Evaluation of the Coefficients $c_\alpha$:** The coefficients $c_\alpha$ of the quadriexcitations are calculated from the expression:

$$c_\alpha^Q = \sum_{(i,j)} c_i c_j \tag{18}$$

where the pair of indices $(i, j)$ runs on all di-excitations of $\phi_0$ that generates $\phi_\alpha$. The coefficients of tri-excitations are calculated like in the method TD1 (Equation (15)) for the contributions *connected or nonfactori-*

*zables* and in according with the expression:

$$c_{\alpha}^{T,U} = \sum_{(i,j)} c_i c_j \tag{19}$$

where $(i, j)$ runs on pairs (mono, diexcitations) of $\phi_0$ that generate $\phi_\alpha$, for the contributions *disconnected or factorizables*. This method is denominated TD2' or MVTD (Mean Value Total Dressing). A variant of this method, denominated TD2'EPV, consists of evaluating the contributions of the *connected* triexcitations in TD1EPV method.

## 4. Projection Methods for Sparse Eigenvalue Problems

In this section, we discuss projection methods for solving the real symmetric standard eigenvalue problem,

$$Ax = \lambda x, \tag{20}$$

where $A \in \mathbb{R}^{n\times n}$, $\lambda \in \mathbb{R}$ (eigenvalue) and $x \in \mathbb{R}^n$ (eigenvector). Projection methods are appropriate when the matrix *A* is very large but its application to a vector is relatively cheap (e.g., it is sparse) and only part of the spectrum is required. If we number the eigenvalues consecutively from left to right, then we might want to compute the first *k* eigenpairs, $(\lambda_i, x_i)$, $i = 1, \cdots, k$, usually with $k \ll n$. The basic principle of projection methods is to find the best approximations to the eigenvectors in a given subspace of small dimension. Let *V* be an $n \times m$ matrix, with $k \le m \ll n$, whose columns $v_i$ constitute an orthonormal basis of a given subspace $\mathcal{V}$, *i.e.*, $V^T V = I_m$ and $\text{span}\{v_1, v_2, \cdots, v_m\} = \mathcal{V}$. Then the eigenvalues of the so-called Rayleigh quotient matrix $T = V^T AV$ approximate eigenvalues of the original matrix. More precisely, if $Ty_i = \theta_i y_i$ then the eigenpair approximations are $\tilde{\lambda}_i = \theta_i$ and $\tilde{x}_i = Vy_i$. These approximate eigenvectors belong to subspace $\mathcal{V}$ and are the best possible approximations in that subspace. For background material on projection methods, the reader is referred to [8] [9].

### 4.1. Restarted Krylov Methods

The quality of the eigenpair approximations $(\tilde{\lambda}_i, \tilde{x}_i)$ depends on how the subspace $\mathcal{V}$ is built. A popular choice is to use the Krylov subspace associated with matrix *A* and a given initial vector $v_1$, where:

$$\mathcal{K}_m(A, v_1) = \text{span}\{v_1, Av_1, A^2 v_1, \cdots, A^{m-1} v_1\} \tag{21}$$

Without loss of generality, in the sequel we will assume that $v_1$ has unit length.

In the case of a symmetric matrix, we can use the Lanczos algorithm to compute an orthogonal basis of the Krylov subspace. This method also provides the projected matrix *T*, which is a symmetric tridiagonal matrix in this case. This computation is efficient and numerically stable, provided that an appropriate methodology is employed to guarantee a good quality of orthogonality among the basis vectors. In brief, the full-orthogonalization Lanczos method computes the basis vectors $v_j$ in sequence, starting from $v_2$. Each vector $v_j$ is the result of orthogonalizing $Av_{j-1}$ with respect to the previous $j-1$ vectors, and then normalizing with respect to the Euclidean norm.

After *m* steps of the Lanczos algorithm, the computed quantities satisfy the relation

$$AV = VT + f_{m+1} e_m^T, \tag{22}$$

where $f_{m+1}$ would yield the next basis vector (after normalization) if the process were to be continued, and $e_m$ is the *m*th coordinate vector.

As the number of columns in *V* grows, eigenvalues of *T* will tend to converge to eigenvalues of *A*. The rate of convergence will be fast provided that the initial vector $v_1$ is rich in the direction of the wanted eigenvectors. However, in practice this is usually not the case and consequently many iterations are likely to be required. This is a serious problem because increasing the number of iterations (*m*) implies a growth in storage requirements and, more importantly, a growth of computational cost per iteration (mainly because orthogonalization is increasingly expensive and also because the cost of computing eigenpairs of *T* becomes non-negligible). A workaround for this is to *restart* the algorithm, that is, stop after *m* iterations and rerun the method with a new $v_1$ computed from the recently obtained spectral approximations. One possible approach is to explicitly compute $v_1$ as a linear combination of a subset of the current approximate eigenvectors. This is called explicit restart.

The main difficulty with explicit restart is how to choose the parameters for building the new starting vector.

A better approach is to do some form of implicit restart, that avoids the need to explicitly compute a new starting vector $v_1$. In the context of Lanczos, one such technique is the so-called thick-restart Lanczos method [10]. In brief, this technique keeps $\ell$ approximate eigenvectors (those corresponding to eigenvalues in the wanted part of the spectrum) and discards the rest, then completes the basis with $m-\ell$ new Lanczos vectors (a typical choice is $\ell = m/2$). Apart from enhancing convergence between restarts, this technique allows the iteration to focus on a certain part of the spectrum, such as the leftmost eigenvalues.

The restart is also a good moment to keep track of already converged eigenvalues, so that they can be deflated. Deflating converged eigenvalues means modifying the iteration so that they do not reappear in the spectrum of *T*. This can be done in different ways, but the most effective one is called locking and consists in extracting the converged eigenvector from the active basis *V* but still consider it in the orthogonalization step. A schematic description of Lanczos with thick-restart and locking is shown below.

1) Set $\tilde{X} = \varnothing$ (already converged eigenvectors)
2) Run *m* steps of Lanczos
3) Restart loop (until *k* eigenpairs have converged)
a) Compute eigenpairs of *T* and check for newly converged eigenpairs
b) Add newly converged eigenvectors to $\tilde{X}$
c) Keep $\ell$ approximate eigenvectors in the basis
d) Run $m-\ell$ steps of Lanczos

Note that the number of columns of $\tilde{X}$ can grow as required. In this way, the iterative eigensolver can be used to compute more than *m* eigenpairs. For instance, if one wants to compute *k* = 8000 eigenpairs of a matrix of order *n* = 50,000, then setting for instance *m* = 10,000 would be too much computational effort for extracting eigenpairs of *T*, whereas with the above scheme it is possible to work with *m* = 300, say, and iterate until all wanted eigenpairs have been retrieved.

## 4.2. The SLEPc Library

SLEPc, the Scalable Library for Eigenvalue Problem Computations [11], is a software package for the solution of large-scale eigenvalue problems on parallel computers. Apart from the standard eigenvalue problem of Equation (20), it also addresses other types of problems such as the generalized eigenvalue problem or the singular value decomposition. SLEPc can work with either real or complex arithmetic, in single or double precision, and it is not restricted to symmetric (Hermitian) problems. It can be used from code written in C, C++, FORTRAN, and Python. SLEPc has been employed successfully in many different application areas such as nuclear engineering [12] or plasma physics [13].

SLEPc provides a collection of eigensolvers, most of which are based on the subspace projection paradigm described in the previous paragraphs. In particular, it includes a robust and efficient parallel implementation of the thick-restart Lanczos method, as well as its more general variant for non-symmetric problems called the Krylov-Schur method [14]. The Lanczos solver includes the possibility to set the *m* parameter described in the previous subsection, thus allowing the computation of a large number of eigenpairs. This feature is missing in other software such as ARPACK [15]. In addition to Krylov solvers, other methods such as Davidson-type or conjugate-gradient solvers are under development.

SLEPc is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation [16]), a parallel framework for the numerical solution of partial differential equations, whose approach is to encapsulate mathematical algorithms using object-oriented programming techniques in order to be able to manage the complexity of efficient numerical message-passing codes. All the PETSc software is freely available and used around the world in many application areas. PETSc is object-oriented in the sense that all the code is built around a set of data structures and algorithmic objects. The application programmer works directly with these objects rather than concentrating on the underlying data structures. The three basic abstract data objects are index sets, vectors and matrices. Built on top of this foundation are various classes of solver objects, including linear, non-linear and time-stepping solvers. SLEPc inherits all the good properties of PETSc, including portability to a wide range of parallel platforms, scalability to a large number of processors, and run-time flexibility giving full control over the solution process (one can for instance specify the solver at run time, or change relevant parameters such as the tolerance or the size of the subspace basis).

The solvers in PETSc and SLEPc have a data-structure neutral implementation. This means that the computation can be done with different matrix storage formats, and also even with a matrix that is not stored explicitly (this requires some user-defined operations such as the matrix-vector product). By default, a matrix in PETSc is stored in a parallel compressed-row sparse format (called aij), where each processor stores a subset of rows. Other formats include the symmetric variant (sbaij), where only the upper triangular part is stored, as well as the dense storage (both sequential and parallel).

## 5. Older Results

In the past we made the TQ2 parallelization [17] using the PVM library and master and slave strategy, with one processor making only the data distribution and communications. This work was a very important advance, but nowadays it is not powerful for big molecules.

Trial calculations with the developed parallel code were carried out on the following platforms:

- Cluster of IBM RISC System/6000 370 workstations with AIX 3.2.5 Operating System ans xlf 2.3 FORTRAN compilation.
- SP1 computer (first generation HPS) with Thin nodes. AIX Version 3.2.4 and Fortran-90 compiler xlf v3.2. The message exchange was performed with the standard PVM library (version 3.3.5) and TCP/IP on the switch.
- SGI Origin 2000 with 64 processors R10000/195 Mhz nodes. The application was compiled with native SGI PVM library.

We have chosen as trial molecule the HF molecule with a very large basis set, a contracted basis set [5s, 4p, 3d, 2f, 1g] for the F atom and [4s, 3p, 2d, 1f] for the H atom.

**Tables 1-3** show the results in terms of Execution Time, Speedup and Efficiency on the three platforms. Results are presented too in **Figures 2-4**.

Obviously, the use of master-slave strategy (with one node only make the job distribution) is a not correct solution. Now, we are using the actual parallel libraries PETSc [16] and SLEPc [11] and a MPI system, can make

**Table 1.** Execution time (seconds) in different systems.

| Processors | FDDI | SP1 | O2000 |
|---|---|---|---|
| 2 | 712.93 | 638.96 | 148.90 |
| 4 | 248.91 | 241.11 | 52.57 |
| 8 | | 120.37 | 26.51 |
| 16 | | | 18.59 |

**Table 2.** Speedup in different systems.

| Processors | FDDI | SP1 | O2000 |
|---|---|---|---|
| 2 | 0.96 | 0.95 | 0.99 |
| 4 | 2.74 | 2.70 | 2.80 |
| 8 | | 5.41 | 5.56 |
| 16 | | | 7.93 |

**Table 3.** Efficiency in different systems.

| Processors | FDDI | SP1 | O2000 |
|---|---|---|---|
| 2 | 0.48 | 0.47 | 0.49 |
| 4 | 0.68 | 0.67 | 0.60 |
| 8 | | 0.67 | 0.69 |
| 16 | | | 0.49 |

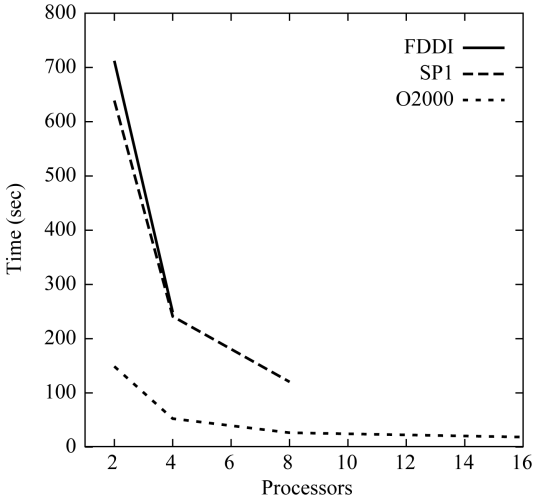


**Figure 2.** Execution time for HF.

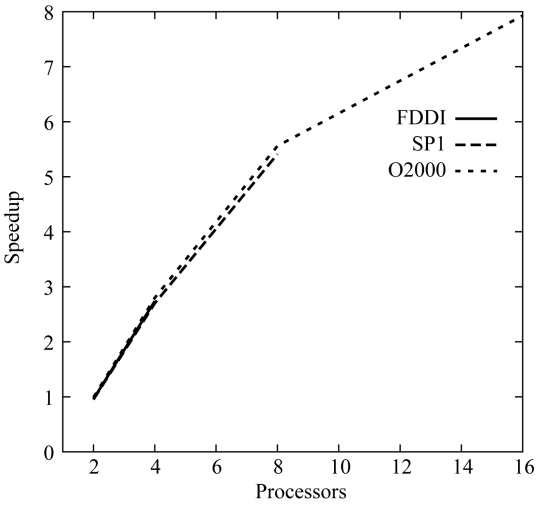


**Figure 3.** Speedup for HF.

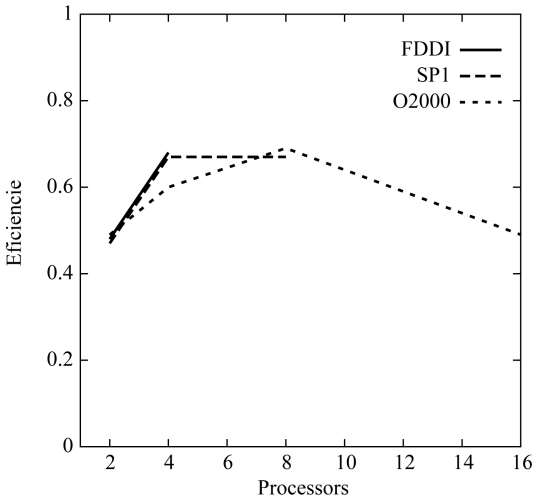


**Figure 4.** Eficiencie time for HF.

and optimal parallelization faster and best built that the older.

## 6. New Parallelization Strategy

In this section, we provide some details about the implementation of the codes, by making use of the parallel capabilities of the SLEPc and PETSc frameworks. We place especial emphasis on optimization issues, such as efficient memory management and reduction of memory requirements.

### 6.1. How SLEPc Eigensolvers Are Parallelized

We start by describing the parallelization strategy used by SLEPc eigensolvers. Both PETSc and SLEPc are oriented to large-scale computation on distributed memory parallel computers with a message-passing paradigm (with the MPI standard). As mentioned before, matrices in PETSc are generally stored by blocks of rows. Vectors also follow the same data distribution, so every processor owns a continuous chunk of the vector elements. According to the description of the Lanczos algorithm in section 4, we can discuss the parallelization of the main operations:

- Matrix-vector product, $Av_{j-1}$. In PETSc, this operation is implemented with mesh-based computations in mind, so that it is particularly efficient in finite-element applications, for example.
- Vector operations that require global communication, such as orthogonalization, inner product and norm. Global communication should be avoided whenever possible, and SLEPc implementations are carefully developed with this issue in mind [18].
- Trivially parallelizable vector operations, such as addition.
- Operations on the small projected matrix, *T*. These operations are carried out sequentially, in a replicated way (all processors perform the same computation). This is usually benign as to parallel performance, provided that the size of the projected problem is not large. This is one of the reasons why using a small value of *m* compared to *k* can be beneficial. The other reason is memory consumption as described next.

In order to determine the memory requirements of the eigensolver, we have to focus on Equation (22). We assume that matrix *A* is of dimension *n*. Apart from the matrix *A* (whose storage will be discussed later) and minor workspace, the memory requirements can be summarized as follows:

- Storage of basis vectors *V* and eigenvectors *X*: the number of columns of *V* grow up to the maximum, *m* (mpd in SLEPc's terminology), and there is also need to store the eigenvectors as they converge, up to the requested number, *k* (nev in SLEPc's terminology). So we need to store at least $m+k$ vectors, that is $n_{\text{local}}(m+k)$ floating-point numbers per processor (assuming the local dimension is $n_{\text{local}} = n/p$, and *p* is the number of processors).
- Storage of the projected matrix: for this we need two square arrays, one for *T* and the other for its eigenvectors, both of dimension $m \times m$. This memory is replicated in all processors, not distributed. This amount of memory is negligible except in the case that *m* is too large.

As an example, suppose we have a problem of order *n* = 80,000 and we want to solve it with *p* = 8 processors, computing *k* = 1000 eigenvalues with *m* = 500. The local part of the vectors would be $n_{\text{local}} = 10,000$. The memory in each processor would be:

- Basis vectors and eigenvectors: $n_{\text{local}}(m+k) = 15,000,000$
- Projected problem: $2m^2 = 500,000$
- Total: 15,500,000 floating-point numbers, that is roughly 120 Mbytes (in double precision).

We mentioned in passing that all the code is prepared to work correctly in single precision arithmetic. In the following, we use double precision throughout.

### 6.2. Matrix Storage

Together with the sparse storage, distributing the matrix across several processes makes it possible to increase the size of the molecular cluster under study, which is crucial for analyzing problems of real scientific interest. However, it is possible to enhance the properties related to parallelization by using an appropriate ordering, as will be discussed below. Before that, we treat two important issues: memory preallocation and exploitation of symmetry.

Matrix preallocation is necessary because PETSc uses a dynamic memory scheme for flexible storage of ma-

trices (the used memory grows as new nonzero elements are added). Since allocation of memory is time-consuming, this scheme is very inefficient especially for very large matrices with many nonzero elements. The solution is to preallocate, that is, to do a priori estimation of the number of nonzero elements and reserve a sufficiently large chunk of memory for them. Sometimes, it is sufficient to estimate nonzero elements very roughly for preallocation. In our case, we have the CSR matrix storage and know the non zeros elements.

Regarding the symmetry of the matrix, we have three options:

- aijfull: Compute all matrix elements, without taking symmetry into account.
- aijhalf: Compute half of the elements only (e.g., the upper triangular part), but store each element twice (both in the original position and the symmetric one). The drawback of this option is that it requires explicit interprocessor communication during matrix assembly.
- sbaij: Compute half of the elements only and use a special symmetric storage format.

For a matrix with $n_{nz}$ nonzero elements, the total storage requirement is about $n_{nz}$ floating point numbers and $n+n_{nz}$ integers. In the case of the special sbaij format, the memory usage is reduced roughly by half. By default, we use the symmetric storage format, because it saves a lot of memory and it is the best one in terms of matrix generation time.

Although the number of matrix rows assigned to each process is roughly the same, the different number and position of nonzero elements in each row can lead to load imbalance and excessive communication overhead in the matrix-vector product. For best performance, it may be necessary to perform an appropriate reordering of the unknowns, and permute the matrix accordingly. As it is well known, in this context a good ordering can be computed by partitioning the adjacency graph associated to the nonzero structure of the matrix, where the goal is to obtain *p* partitions of almost equal size while minimizing the edge-cut between partitions. For this aim, we have used ParMETIS [19].

## 7. Code Evaluation

We have carried out a number of numerical experiments in order to validate the correctness of the parallel codes, as well as to assess their parallel efficiency.

The computer system used for the computational experiments is Tirant, an IBM cluster consisting of 256 JS20 blade computing nodes, each of them with two 64-bit PowerPC 970FX processors running at 2.2 GHz, interconnected with a low latency Myrinet network.

### 7.1. Test Cases and Validation of Numerical Results

For the code, the experiments we report in this paper are related to the simulation of HF and HB molecules with a very large basis set, a contracted basis set [5s, 4p, 3d, 2f, 1g] for the F and B atom and [4s, 3p, 2d, 1f] for the H atom. The objective is to calculate the electron correlation energy.

In order to validate the correctness of the computations, we compare the results from the parallel versions with the results from the original sequential code.

### 7.2. Parallel Performance for FH

We now turn our attention to the parallel performance analysis of the overall computation. We will use the terms aij and sbaij to refer to the cases where explicit matrix *axpy* operations are used with the two matrix storage strategies considered in Section 6.2.

In terms of actual response time, we can observe that sbaij have the best time and speedup. We can see a very good reduction in execution time and a good parallel strategy in the code.

In **Tables 4-6** and **Figures 5-7** we can see the results obtained.

### 7.3. Parallel Performance for HB

We now turn our attention to the parallel performance analysis of the overall computation. We will use the terms aij and sbaij to refer to the cases where explicit matrix *axpy* operations are used with the two matrix storage strategies considered in Section 6.2.

In this case, we can observe that sbaij have the best time and speedup too. We can see a very good reduction in execution time and a good parallel strategy in the code.

**Table 4.** Execution time (in seconds) corresponding to the diagonalization of HF molecule with increasing number of nodes.

| Processors | aij | sbaij |
|---|---|---|
| 1 | 75.6 | 70.6 |
| 2 | 43.4 | 37.8 |
| 4 | 24.6 | 18.4 |
| 8 | 13.7 | 9.5 |
| 16 | 7.3 | 4.8 |

**Table 5.** Speedup corresponding to diagonalization of HF molecule with increasing number of nodes.

| Processors | aij | sbaij |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1.74 | 1.87 |
| 4 | 3.07 | 3.84 |
| 8 | 5.52 | 7.43 |
| 16 | 10.36 | 14.71 |

**Table 6.** Eficiencie corresponding to the diagonalization of HF molecule with increasing number of nodes.

| Processors | aij | sbaij |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 0.87 | 0.93 |
| 4 | 0.77 | 0.96 |
| 8 | 0.69 | 0.93 |
| 16 | 0.65 | 0.92 |

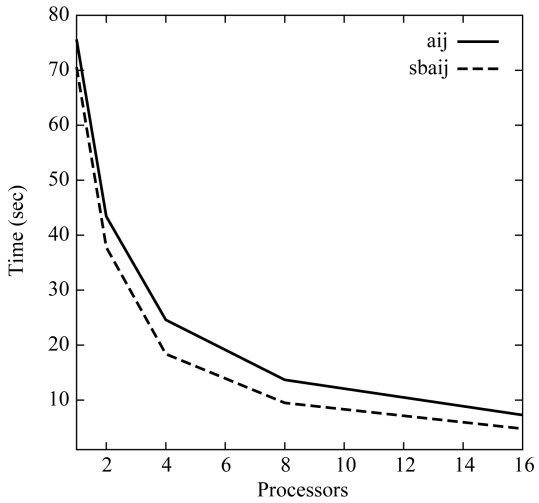


**Figure 5.** Execution time (seconds) for HF with SLEPc.

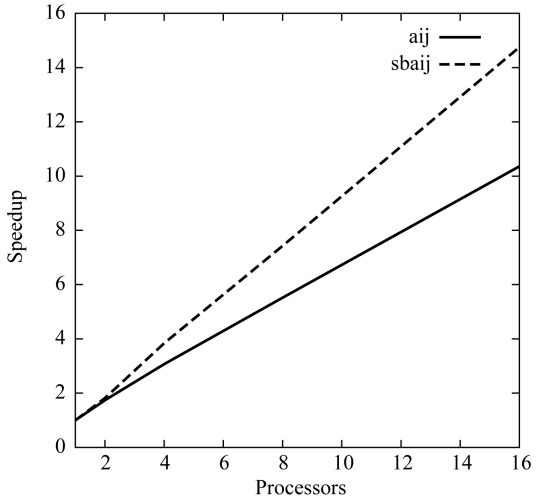


**Figure 6.** Speedup for HF with SLEPc.

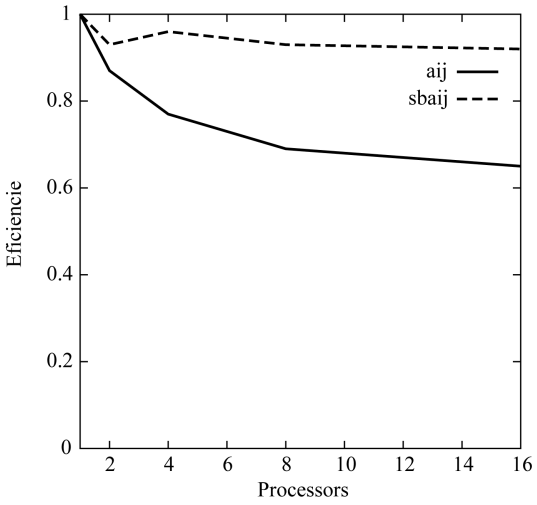


**Figure 7.** Eficiencie time for HF with SLEPc.

In **Tables 7-9** and **Figures 8-10** we can see the results obtained.

## 8. Applications of the New Method

This new method for parallel calculation in computational chemistry has a lot of very important applications:

- Parallelization of the molecular magnetic susceptibility
- Parallelization of the anisotropic and isotropic magnetism in molecules
- Parallelization of energy in every spin in a big molecule
- Computation of the molecules never calculated with sequential codes

All these applications need a very strong computational effort for your calculation and the parallel use of PETSc and SLEPc offer us the possibility of resolve a very big new problem.

## 9. Conclusions and Future Work

In this paper we have presented the TQ2 program parallelization. The critical part of the computation, diagonalization large-scale sparse matrices, is carried out by means of the SLEPc library. We have evaluated the developed code with one moderate-sized test case, which provides us with feedback about the correctness of the new programs, and gives us an idea of the scalability to large number of processors. The main conclusions that we

**Table 7.** Execution time (in seconds) corresponding to the diagonalization of HB molecule with increasing number of nodes.

| Processors | aij | sbaij |
|---|---|---|
| 1 | 70.2 | 65.3 |
| 2 | 41.7 | 38.8 |
| 4 | 22.2 | 19.7 |
| 8 | 11.2 | 9.6 |
| 16 | 6.3 | 5.8 |

**Table 8.** Speedup corresponding to diagonalization of HB molecule with increasing number of nodes.

| Processors | aij | sbaij |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1.68 | 1.69 |
| 4 | 3.16 | 3.31 |
| 8 | 6.27 | 6.8 |
| 16 | 12.32 | 12.8 |

**Table 9.** Eficiencie corresponding to the diagonalization of HB molecule with increasing number of nodes.

| Processors | aij | sbaij |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 0.84 | 0.85 |
| 4 | 0.79 | 0.83 |
| 8 | 0.78 | 0.82 |
| 16 | 0.77 | 0.80 |

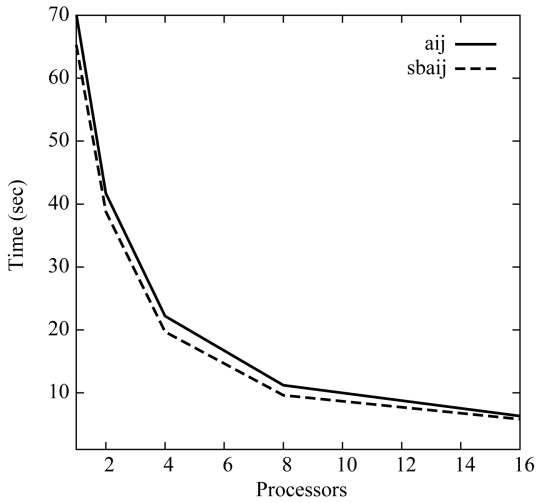


**Figure 8.** Execution time (seconds) for HB with SLEPc.

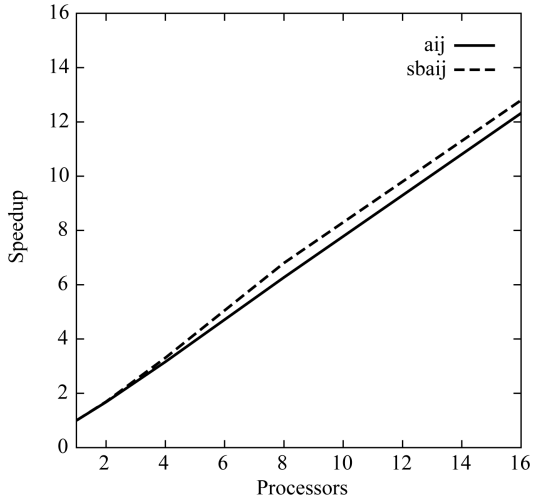


**Figure 9.** Speedup for HB with SLEPc.

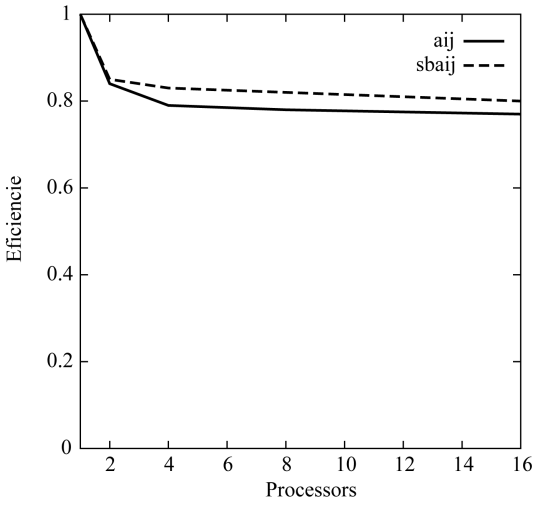


**Figure 10.** Eficiencie time for HB with SLEPc.

can draw are the following:

- The parallelization of the program allows us the drastic reduction of the response time of the calculation, compared to the original sequential programs and the older parallel version. This is a great advantage, e.g., in the solution of moderate-size problems with few processors. More importantly, parallelization will make it possible to solve much larger problems, those with real scientific interest, which would otherwise be impossible to address due to memory limitations or lack of computational power.
- The use of state-of-the-art iterative eigensolvers available in SLEPc, which are intended for large-scale sparse matrices, is a major improvement over the original codes. These methods allow for the computation of just the part of the spectrum of interest, thus saving a lot of computational effort, and can address huge problem sizes. In this particular application, the SLEPc solvers have proved to be very robust and efficient, with a very fast convergence.
- We have evaluated different matrix storage schemes and several strategies for handling matrices of the form $A+\gamma B$. We have found that symmetric storage very effective. The evaluation has also provided us with very valuable information concerning memory requirements, which will allow us to make accurate estimates of memory consumption when solving huge problems.
- We have a very good results using SLEPc, a useful tool for chemical codes.

Once we have demonstrated the viability of our approach, the next step is to use the TQ2 program in produc-

tion mode, that is, to address challenging problems with real scientific interest. Some preliminary results with large molecular clusters are very encouraging and we foresee to achieve some significant breakthroughs in this area in a short term.

We also plan to continue improving the programs. In the next months, we will evaluate some new eigensolvers that are under development in SLEPc, in particular the Davidson-type methods.

A broader view of the future work is to apply the same methodology to other codes and applications in related areas. The use of the SLEPc library opens the possibility of parallelizing various programs in computational chemistry whose scientific advantages are very relevant. This paradigm shift is going to allow us to solve problems that until now were impossible to address, since sequential computation is inviable due to the amount of time and memory required.

## Acknowledgements

The authors are grateful for the computing resources provided by the Spanish Supercomputing Network (RES). The simulations were carried out on the supercomputer Tirant at Universitat de València.

## References

[1] McWeeny (1992) Methods in Computational Molecular Physics. Plenum Press, London.

[2] Maynau, D. and Heully, J.L. (1993) Second-Order Perturbation on a SDCI Calculation. *Chemical Physics Letters*, **211**, 625-630. http://dx.doi.org/10.1016/0009-2614(93)80154-H

[3] Sánchez-Marn, J., Maynau, D. and Malrieu, J.P. (1993) *Theoretica Chimica Acta*, **87**.

[4] McWeeny (1987) *Ab Initio* Methods in Quantum Chemistry. Part I and Part II, John Willey and Sons, New York.

[5] Dupuis, M., Rys, J. and King, H.F.J. (1976) *Chemical Physics*, **65**, 111.

[6] Malrieu, J.P., Durand, P. and Daudey, J.P. (1985) Intermediate Hamiltonians as a New Class of Effective Hamiltonians. *Journal of Physics A*: *Mathematical and General*, **18**, 809. http://dx.doi.org/10.1088/0305-4470/18/5/014

[7] Malrieu, J.P., Nebot-Gil, I. and Sánchez-Marn, J. (1994) Elementary Presentation of Self-Consistent Intermediate Hamiltonians and Proposal of Two Totally Dressed Singles and Doubles Configuration Interaction Methods. *The Journal of Chemical Physics*, **100**, 1440. http://dx.doi.org/10.1063/1.466622

[8] Parlett, B.N. (1998) The Symmetric Eigenvalue Problem. Prentice-Hall, Englewood Cliffs.

[9] Bai, Z., Demmel, J., Dongarra, J., Ruhe, A. and van der Vorst, H. (Eds.) (2000) Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. Society for Industrial and Applied Mathematics, Philadelphia. http://dx.doi.org/10.1137/1.9780898719581

[10] Wu, K. and Simon, H. (2000) Thick-Restart Lanczos Method for Large Symmetric Eigenvalue Problems. *SIAM Journal on Matrix Analysis and Applications*, **22**, 602-616. http://dx.doi.org/10.1137/S0895479898334605

[11] Hernandez, V., Roman, J.E. and Vidal, V. (2005) SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems. *ACM Transactions on Mathematical Software*, **31**, 351-362. http://dx.doi.org/10.1145/1089014.1089019

[12] Gilbert, D., Roman, J.E., Garland, W.J. and Poehlman, W.F.S. (2008) Simulating Control Rod and Fuel Assembly Motion Using Moving Meshes. *Annals of Nuclear Energy*, **35**, 291-303. http://dx.doi.org/10.1016/j.anucene.2007.06.007

[13] Roman, J.E., Kammerer, M., Merz, F. and Jenko, F. (2010) Fast Eigenvalue Calculations in a Massively Parallel Plasma Turbulence Code. *Parallel Computing*, **36**, 339-358. http://dx.doi.org/10.1016/j.parco.2009.12.001

[14] Stewart, G.W. (2001) A Krylov-Schur Algorithm for Large Eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, **23**, 601-614. http://dx.doi.org/10.1137/S0895479800371529

[15] Lehouc, R.B., Sorensen, D.C. and Yang, C. (1998) ARPACK Users' Guide, Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods. Society for Industrial and Applied Mathematics, Philadelphia. http://dx.doi.org/10.1137/1.9780898719628

[16] Balay, S., Buschelman, K., Eijkhout, V., Gropp, W., Kaushik, D., Knepley, M., McInnes, L.C., Smith, B. and Zhang, H. (2008) PETSc Users Manual. Tech. Rep. ANL-95/11—Revision 3.0.0, Argonne National Laboratory.

[17] Ramos, E., Daz, W. and Cerverón, V. (1998) Paralelización bajo PVM del cálculo de la energa de correlación electrónica mediante revestimiento de matrices, IX Jornadas de Paralelismo. Universidad del País Vasco, San Sebastián.

[18] Hernandez, V., Roman, J.E. and Tomas, A. (2007) Parallel Arnoldi Eigensolvers with Enhanced Scalability via Global Communications Rearrangement. *Parallel Computing*, **33**, 521-540.

[19] Karypis, G. and Kumar, V. (1999) Parallel Multilevel k-Way Partitioning Scheme for Irregular Graphs. *SIAM Review*, **41**, 278-300. http://dx.doi.org/10.1137/S0036144598334138