JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

Full Parallel Implementation of an All-Electron Four-Component Dirac–Kohn–Sham Program

Sergio Rampino,*[†] Leonardo Belpassi,*[†] Francesco Tarantelli,[‡] and Loriano Storchi*[§]

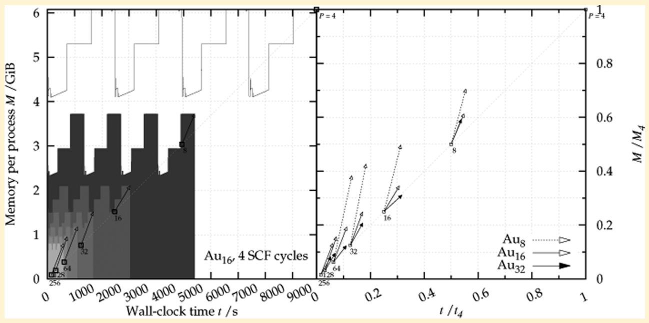
[†]Istituto di Scienze e Tecnologie Molecolari, Consiglio Nazionale delle Ricerche c/o Dipartimento di Chimica, Biologia e Biotecnologie, Università degli Studi di Perugia, Via Elce di Sotto 8, 06123 Perugia, Italia

[‡]Dipartimento di Chimica, Biologia e Biotecnologie, Università degli Studi di Perugia, Via Elce di Sotto 8, 06123 Perugia, Italia

[§]Dipartimento di Farmacia, Università degli Studi 'G. D'Annunzio', Via dei Vestini 31, 66100 Chieti, Italia

Supporting Information

ABSTRACT: A full distributed-memory implementation of the Dirac–Kohn–Sham (DKS) module of the program BERTHA (Belpassi et al., *Phys. Chem. Chem. Phys.* **2011**, *13*, 12368–12394) is presented, where the self-consistent field (SCF) procedure is replicated on all the parallel processes, each process working on subsets of the global matrices. The key feature of the implementation is an efficient procedure for switching between two matrix distribution schemes, one (integral-driven) optimal for the parallel computation of the matrix elements and another (block-cyclic) optimal for the parallel linear algebra operations. This approach, making both CPU-time and memory scalable with the number of processors used, virtually overcomes at once both time and memory barriers associated with DKS calculations. Performance, portability, and numerical stability of the code are illustrated on the basis of test calculations on three gold clusters of increasing size, an organometallic compound, and a perovskite model. The calculations are performed on a Beowulf and a BlueGene/Q system.



1. INTRODUCTION

The impact of relativity on chemistry has shown over the past years to be of great importance, especially when heavy elements are involved.[1−4] For a quantitative description of the electronic structure and properties of organometallic compounds, clusters, and nanoscale materials containing heavy elements, a proper inclusion of relativistic effects in the relevant equations is mandatory. The most rigorous way to introduce relativity in the modeling of molecular systems is to adopt the four-component formalism derived from the Dirac equation.[5] Electron correlation effects, on the other hand, also play a crucial role in determining the electronic structure and properties of molecules. The method of choice in this case is Density Functional Theory (DFT), which is normally cast in the form of the independent-particle Kohn–Sham approach where all of the exchange-correlation effects are expressed implicitly as a functional of the electron density or, more generally, of the charge-current density.[6,7]

Relativistic effects and electron correlation effects are in general not additive and should be treated on the same footing. A robust and efficient approach capable of accounting for both, is based on the combination of the rigorous four-component relativistic formalism with DFT. The resulting model, referred to as the Dirac–Kohn–Sham (DKS) model, has been introduced several years ago[8] and significant progress has since been achieved in both theory and implementation.[9−16] We recently reviewed both aspects[17] with a focus on our own implementation in the program BERTHA.

Full four-component DKS calculations have an intrinsically larger computational cost (both in terms of computing time and required memory) than analogous nonrelativistic approaches. This arises from the complex representation required for the involved matrices and the intrinsically larger basis sets, as a consequence of the four-component structure of the DKS equation. Regarding computing time, however, only a larger prefactor in the scaling with the number of particles or basis set size shows up, not a more unfavorable power law. A leap forward in the applicability of the DKS method was achieved thanks to the adoption of electron-density fitting approaches,[18,19] that are already routinely used in the nonrelativistic context. In our own implementation of BERTHA, for example, where density fitting techniques were adopted both for the Coulomb[20] and for the exchange-correlation[21] problem, besides reducing the formal scaling from $O(N^4)$ to $O(N^3)$, we could achieve a dramatic reduction of the prefactor. This allowed for integral evaluation speedups of up to several orders of magnitude without significant loss in accuracy. On the other side, quasi-relativistic approaches have been proposed (see for example ref 22 and references therein) to improve the efficiency of DKS calculations, such as the quasi-4-component

Received: June 9, 2014
Published: July 24, 2014

approach of refs 23 and 24 that halves the DKS matrix size without significantly compromising the accuracy.

Within the rigorous full four-component framework, however, despite the mentioned algorithmic advances, we had to resort very early to parallel computing.[25] The huge size of basis sets required for an accurate all-electron treatment of many heavy atoms made computational requirements simply unaffordable on a typical computer. The target was, of course, a parallel implementation allowing both execution time and memory usage scale with number of processors. This is nowadays even more appropriate, as the current architectural trend seems to evolve toward many cores and massively parallel systems where the computational resources are made of many compute units (cores) having each a relatively small amount of memory (see for instance Nvidia's CUDA,[26] Intel Many Integrated Core Architecture,[27] or the IBM BlueGene project[28]). Similar efforts, indeed, have been recently undertaken by other teams developing electronic structure programs (see for example ref 29).

In a first "memory oriented" parallel implementation of BERTHA, we adopted a sort of master-slave paradigm, which, though very satisfactory in the DKS matrix construction phase, only partially solved the memory bottleneck problem.[30] The effective memory distribution was achieved thanks to a fast but specific feature of the SGI Altix 4700 system where we developed and tested the implementation. The way toward a fully distributed implementation was pointed out in a more recent work,[31] where we combined, using the Message Passing Interface (MPI),[32] our distributed memory algorithm to construct the Coulomb and exchange-correlation matrices with the ScaLAPACK library[33] routines performing parallel linear algebra operations on distributed matrices. That implementation still featured a serial portion of code associated with the computationally light (compared to the matrix build and linear algebra operations) density fitting step. This imposed two big limitations to the efficiency of the code: (i) due to Amdahl's law,[34] the density fitting step turned out to be the limiting one when an (already moderately) high number of processes (>64) was employed; (ii) two of the large DKS matrices still had to fully reside on the single process in charge of carrying out the density fitting step.

With the present work, the last mile for a complete distributed memory implementation is finally reached as a result of (i) the systematic removal of any serial portion of code and (ii) the extension and generalization of the distributed memory approach already devised for the Coulomb and exchange-correlation matrix to all of the matrices involved. In this new scheme, all of the required potentially huge complex matrices are, at any time during the execution, apportioned among the parallel processes and the original SCF procedure is replicated entirely on all of the processes in a "symmetric" fashion, so that every process executes the same instructions on subsets of the global matrices. The matrix distribution scheme is such that, at each time step, data are always distributed among the various processes to obtain the best performance with respect to the actual parallel algorithm adopted. Many modern quantum chemistry programs adopt a method to simulate shared memory for distributed memory computers (e.g., ADF[35] uses GlobalArray,[36] GAMESS-US[37] uses the Distributed Data Interface[38] library). We designed an ad hoc solution for this. This new full parallel implementation, allowing for the simultaneous scaling of both memory and time requirements, considerably extends the range of applicability of the costly four-component treatment.

The SCF procedure of the DKS module of BERTHA is outlined in section 2. Details on its parallel implementation are given in section 3. Performance is discussed in section 4. Conclusions are drawn in section 5.

2. THE SCF PROCEDURE

The program BERTHA has been described in detail elsewhere.[17,20,21,25,39−43] We briefly review here its SCF procedure for a DKS calculation in order to point out the steps and matrices that have been targeted in the present parallel implementation. The reader is referred to the original papers for a complete information.

The DKS equation using atomic units and including only the longitudinal interactions reads

$$\left[c\begin{pmatrix}0 & \boldsymbol{\sigma}\\ \boldsymbol{\sigma} & 0\end{pmatrix}\boldsymbol{p} + \begin{pmatrix}\boldsymbol{I} & 0\\ 0 & -\boldsymbol{I}\end{pmatrix}c^2 + v^{(1)}(\boldsymbol{r})\right]\boldsymbol{\psi}_i(\boldsymbol{r}) = \varepsilon_i\boldsymbol{\psi}_i(\boldsymbol{r}) \tag{1}$$

where c is the speed of light in vacuum, $\boldsymbol{p}$ is the electron four-momentum, $\boldsymbol{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$ with σ_q being a 2×2 Pauli spin matrix and $\boldsymbol{I}$ is a 2×2 identity matrix. As in the nonrelativistic context, the diagonal potential operator is made up of a nuclear potential term $v_{\mathrm{N}}(\boldsymbol{r})$, a Coulomb interaction term $v_{\mathrm{H}}^{(1)}[\rho(\boldsymbol{r})]$ and an exchange-correlation term $v_{\mathrm{xc}}^{(1)}[\rho(\boldsymbol{r})]$:

$$v^{(1)}(\boldsymbol{r}) = v_{\mathrm{N}}(\boldsymbol{r}) + v_{\mathrm{H}}^{(1)}[\rho(\boldsymbol{r})] + v_{\mathrm{XC}}^{(1)}[\rho(\boldsymbol{r})] \tag{2}$$

the last two terms being a functional of the relativistic electronic density $\rho(\boldsymbol{r})$. (The genuine relativistic exchange-correlation functionals should depend on the relativistic four-current.[6,7,44] However, research activity on these functionals is still at an early stage and mostly limited to the nonrelativistic framework.)

The four-spinor solution of eq 1 is expressed in BERTHA as a linear combination of $2N$ G-spinor basis functions $M_\mu^{(\mathrm{T})}(\boldsymbol{r})$ with T being either L or S

$$\boldsymbol{\psi}_i(\boldsymbol{r}) = \begin{bmatrix} \displaystyle\sum_{\mu=1}^{N} c_{\mu i}^{(\mathrm{L})} M_\mu^{(\mathrm{L})}(\boldsymbol{r}) \\ \displaystyle i\sum_{\mu=1}^{N} c_{\mu i}^{(\mathrm{S})} M_\mu^{(\mathrm{S})}(\boldsymbol{r}) \end{bmatrix} \tag{3}$$

where L identifies the so-called "large" component, S the "small" component, and $c_{\mu i}^{(\mathrm{T})}$ are coefficients to be determined. The G-spinors $M_\mu^{(\mathrm{T})}(\boldsymbol{r})$ are Gaussian-based two-component objects labeled by the collective index μ, which maps univocally onto the set of parameters (Gaussian center and exponent, fine-structure quantum number and magnetic quantum number) necessary to completely characterize the functions (see ref 41 for details on the advantages of such basis). The matrix representation $\boldsymbol{H}_{\mathrm{DKS}}$ of the DKS operator in the G-spinor basis is

$$\begin{bmatrix} \boldsymbol{v}^{(\mathrm{LL})} + \boldsymbol{J}^{(\mathrm{LL})} + \boldsymbol{K}^{(\mathrm{LL})} + mc^2\boldsymbol{S}^{(\mathrm{LL})} & c\boldsymbol{\Pi}^{(\mathrm{LS})} \\ c\boldsymbol{\Pi}^{(\mathrm{SL})} & \boldsymbol{v}^{(\mathrm{SS})} + \boldsymbol{J}^{(\mathrm{SS})} + \boldsymbol{K}^{(\mathrm{SS})} - mc^2\boldsymbol{S}^{(\mathrm{SS})} \end{bmatrix} \tag{4}$$

and the associated eigenvalue equation reads

$$\boldsymbol{H}_{\text{DKS}}\begin{bmatrix}\boldsymbol{c}^{(\text{L})}\\ \boldsymbol{c}^{(\text{S})}\end{bmatrix} = E\begin{bmatrix}\boldsymbol{S}^{(\text{LL})} & 0\\ 0 & \boldsymbol{S}^{(\text{SS})}\end{bmatrix}\begin{bmatrix}\boldsymbol{c}^{(\text{L})}\\ \boldsymbol{c}^{(\text{S})}\end{bmatrix} \tag{5}$$

where $\boldsymbol{c}^{(\text{T})}$ are the spinor expansion vectors of eq 3 and the $\boldsymbol{v}$, $\boldsymbol{J}$, $\boldsymbol{K}$, $\boldsymbol{S}$ and $\boldsymbol{\Pi}$ matrices are the basis representations of the nuclear, Coulomb, and exchange-correlation potentials, the overlap matrix and the matrix of the kinetic energy operator, respectively. (Where the T superscript is dropped, we refer to the whole matrix composed of the LL, LS, LS, and SS blocks.) Their matrix elements are defined by

$$v_{\mu\nu}^{(\text{TT})} = \int v_{\text{N}}(\boldsymbol{r})\rho_{\mu\nu}^{\text{TT}}(\boldsymbol{r})\text{d}\boldsymbol{r} \tag{6}$$

$$J_{\mu\nu}^{(\text{TT})} = \int v_{\text{H}}^{(1)}[\rho(\boldsymbol{r})]\rho_{\mu\nu}^{\text{TT}}(\boldsymbol{r})\text{d}\boldsymbol{r} \tag{7}$$

$$K_{\mu\nu}^{(\text{TT})} = \int v_{\text{XC}}^{(1)}[\rho(\boldsymbol{r})]\rho_{\mu\nu}^{\text{TT}}(\boldsymbol{r})\text{d}\boldsymbol{r} \tag{8}$$

$$S_{\mu\nu}^{(\text{TT})} = \int \rho_{\mu\nu}^{\text{TT}}(\boldsymbol{r})\text{d}\boldsymbol{r} \tag{9}$$

$$\Pi_{\mu\nu}^{(\text{TT}')} = \int M_{\mu}^{(\text{T})\dagger}(\boldsymbol{r})(\boldsymbol{\sigma}\cdot\boldsymbol{p})M_{\nu}^{(\text{T}')}(\boldsymbol{r})\text{d}\boldsymbol{r} \tag{10}$$

with $\rho_{\mu\nu}^{(\text{TT})}(\boldsymbol{r})$ being G-spinor overlap densities.

The relativistic electronic density $\rho(\boldsymbol{r})$ (figuring in eqs 7 and 8) is readily evaluated as

$$\rho(\boldsymbol{r}) = \sum_{\text{T}}\sum_{\mu\nu} D_{\mu\nu}^{(\text{TT})}\rho_{\mu\nu}^{(\text{TT})}(\boldsymbol{r}) \tag{11}$$

with

$$D_{\mu\nu}^{(\text{TT}')} = \sum_{i} c_{\mu i}^{(\text{T})*}c_{\nu i}^{(\text{T}')} \tag{12}$$

where the sum runs over the occupied positive-energy states. The matrix $\boldsymbol{H}_{\text{DKS}}$ depends, because of $\boldsymbol{J}$ and $\boldsymbol{K}$, on the canonical spinor-orbitals produced by its diagonalization, so that the solution $\boldsymbol{c}$ must be obtained recursively to self-consistence. As in the nonrelativistic context, once a guess density has been provided (usually cast as a superposition of atomic densities), the problem formally reduces to the evaluation of the integrals in eqs 6−10 for the assembling of $\boldsymbol{H}_{\text{DKS}}$ (eq 4) and the iterative solution of the eigenvalue problem (eq 5) with up-to-date J and K integrals at each cycle.

As mentioned in the Introduction, we take advantage of density fitting techniques for an efficient evaluation of the Coulomb $\boldsymbol{J}$ and exchange-correlation $\boldsymbol{K}$ matrices. In the density fitting approach implemented in BERTHA and based on the Coulomb metric,[20] the relativistic electronic density is expanded in a set of N_{aux} auxiliary atom-centered functions:

$$\tilde{\rho} = \sum_{t=1}^{N_{\text{aux}}} d_t f_t(\boldsymbol{r}) \tag{13}$$

with the f_s functions chosen to be Hermite Gaussian-Type Functions (HGTF, the advantages of such choice are discussed in ref 20). This allows for an efficient construction of the $\boldsymbol{J}$ and $\boldsymbol{K}$ matrices in a single step[21] through the evaluation of the 3-center two-electron repulsion integrals $I_{s,\mu\nu}^{(\text{TT})} = \langle f_s \| \rho_{\mu\nu}^{(\text{TT})}\rangle$

$$\tilde{J}_{\mu\nu}^{(\text{TT})} + \tilde{K}_{\mu\nu}^{(\text{TT})} = \sum_{t=1}^{N_{\text{aux}}} I_{t,\mu\nu}^{(\text{TT})}(d_t + z_t) \tag{14}$$

once the vectors $\boldsymbol{d}$ and $\boldsymbol{z}$ are made available. These vectors are the solution of two linear equation systems

$$\boldsymbol{Ad} = \boldsymbol{v} \tag{15}$$

$$\boldsymbol{Az} = \boldsymbol{w} \tag{16}$$

where the real, symmetric, and positive-definite matrix $\boldsymbol{A}$ is the representation of the Coulomb interaction in the auxiliary basis, $A_{st} = \langle f_s \| f_t\rangle$. The vector $\boldsymbol{v}$ is the projection of the electrostatic potential on the fitting functions and is efficiently computed via the relativistic analogue of the scalar Hermite density matrix $H_0[\alpha; i, j, k]$ proposed by Almlöf[45,46]

$$v_s = \langle f_s \| \rho\rangle = \sum_{\alpha}\sum_{i,j,k} H_0[\alpha; i, j, k]\langle f_s \| \alpha; i, j, k\rangle \tag{17}$$

with

$$H_0[\alpha; i, j, k] = \sum_{\text{T}}\sum_{\mu,\nu\to\alpha} E_0^{(\text{TT})}[\mu, \nu; i, j, k]D_{\mu\nu}^{(\text{TT})} \tag{18}$$

where the second sum runs over all basis function pairs resulting, by the Gaussian product theorem, in the same origin and exponent labeled by α and the possible values of i, j, and k are univocally determined by the couple (μ, ν). The coefficients $E_0^{(\text{TT})}$, which contain the whole spinor structure, are described in refs 40 and 47. The vector $\boldsymbol{w}$, being the projection of the exchange-correlation potential of the fitted density on the fitting functions

$$w_s = \langle \tilde{v}_{\text{XC}}^{(1)} | f_s\rangle = \int v_{\text{XC}}^{(1)}[\tilde{\rho}(\boldsymbol{r})]f_s(\boldsymbol{r})\text{d}\boldsymbol{r} \tag{19}$$

is instead evaluated numerically using the method described by Becke.[48]

Besides the two small vectors of size N_{aux}, the fitting problem involves a real $N_{\text{aux}} \times N_{\text{aux}}$ matrix. These quantities are anyhow much smaller than the DKS matrices discussed above and have not been targeted in the present parallel implementation. During the SCF procedure, in fact, the "bulk" memory allocation is due to the following $2N \times 2N$ complex Hermitian matrices: (i) the overlap matrix $\boldsymbol{S}$, (ii) the one-electron matrix $\boldsymbol{\Pi} + \boldsymbol{v}$, (iii) the Coulomb plus exchange-correlation matrix $\boldsymbol{J} + \boldsymbol{K}$ (once this matrix has been computed, the associated array is conveniently reused to host the whole $\boldsymbol{H}_{\text{DKS}}$ matrix to be diagonalized), (iv) the matrix $\boldsymbol{c}$ of the eigenvectors, and (v) the density matrix $\boldsymbol{D}$. Accordingly, the memory required for an all-electron run on, say, the Au_{32} gold cluster with double-ζ quality basis set ($2N = 25216$) amounts at least (additional working space is actually required) to 47.35 GiB using double precision. In a previous parallel implementation,[31] we devised a scheme for computing and operating on one of the above matrices, namely, the $\boldsymbol{J} + \boldsymbol{K}$ matrix, in a distributed fashion so that the memory allocation associated with such matrix was equally distributed among the parallel processes. This approach, that will be reviewed in the next section, could easily be extended to the one-electron $\boldsymbol{\Pi} + \boldsymbol{v}$ and overlap $\boldsymbol{S}$ matrices. Still, the serial portion of code associated with the density fitting step (mainly eqs 17 and 18) required the density $\boldsymbol{D}$ and Hermite density $\boldsymbol{H}_0$ (for which a real linearized array dimensioned $2N\cdot 2N$ was used) to reside on one computing node, thus imposing a lower bound on the memory barrier on that node as high as 14.21 GiB for the same Au_{32} case. The main concern of the present work has been to remove this last barrier.

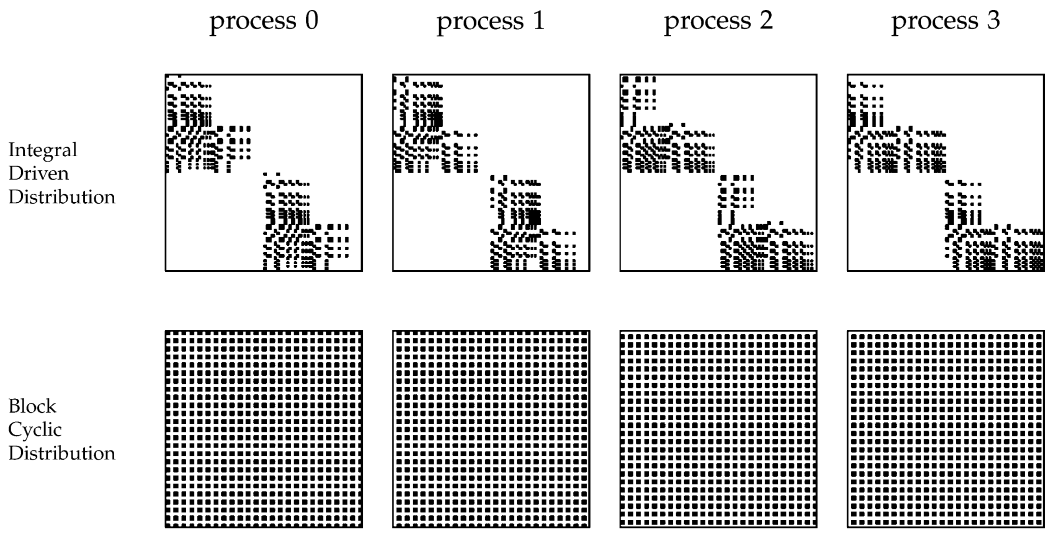


Figure 1. Integral Driven Distribution (IDD) and Block Cyclic Distribution (BCD) for the $\boldsymbol{J} + \boldsymbol{K}$ matrix of the Au_2 system (matrix size 1576) distributed among $P = 4$ processes using a 2×2 process grid and a ScaLAPACK block size of 32. Note that approximately $^1/_4$ of the matrix is computed (IDD panel), with the LS and SL blocks being null (see eq 4) and the rest being worked out by complex conjugation during the mapping to BCD.

3. PARALLEL IMPLEMENTATION

As outlined in the previous section, provided an initial guess density (usually cast as a superposition of atomic densities), the steps involved in a SCF cycle are as follows:

(i) carry out the density fitting through the intermediate array $\boldsymbol{H}_0$ worked out of $\boldsymbol{D}$,

(ii) build up the Coulomb plus exchange-correlation $\boldsymbol{J} + \boldsymbol{K}$ matrix (during the first iteration also the one-electron $\boldsymbol{\Pi} + \boldsymbol{v}$ and overlap $\boldsymbol{S}$ matrices are computed and henceforth conveniently stored in memory, as they do not vary from cycle to cycle),

(iii) assemble $\boldsymbol{H}_{\mathrm{DKS}}$ and solve the eigenvalue problem for a new eigenvector matrix $\boldsymbol{C}$ and density matrix $\boldsymbol{D}$.

3.1. Previous Implementation. Our parallelization efforts toward a memory distributed implementation of the DKS module of BERTHA were prompted by an analysis of the computational cost in terms of CPU-time of the three steps pointed out above for a serial execution. Step i turned out to take only a small fraction of the total CPU time, step ii took about a third, and the rest was entirely due to the linear algebra operations of step iii. We targeted therefore, in order of priority, step iii and step ii, leaving practically unaltered the serial step i.

We took advantage of the widely available and often optimized ScaLAPACK[33] library for carrying out linear algebra operations in parallel on distributed matrices (step iii). We recall here for the reader's convenience the basic features of ScaLAPACK. The P processes of a generic parallel execution are, in ScaLAPACK, mapped onto a $P_r \times P_c$ two-dimensional "process grid" of P_r process rows and P_c process columns ($P_rP_c = P$). A dense matrix is then decomposed into blocks ("ScaLAPACK blocks") of suitable size ("ScaLAPACK block-size"), which are uniformly distributed along each dimension of the process grid (so that every process owns a subset of them) according to a specific—the so-called Block Cyclic Distribution (BCD)[33]—pattern. Once a matrix is distributed among the processes according to the BCD scheme, linear algebra operations are carried out by ScaLAPACK routines in parallel and the output is, again, a BCD distributed matrix.

An efficient parallel construction (step ii) of the Coulomb and exchange-correlation matrix $\boldsymbol{J} + \boldsymbol{K}$ (eq 14) was instead achieved by cyclically assigning to each process the allocation and computation of blocks whose offsets and dimensions depend on the specific structure of the G-spinor matrices. Such block structure, to which we will refer as to Integral Driven Distribution (IDD), is naturally dictated by the grouping of G-spinor basis functions in sets characterized by common origin and angular momentum (see ref 25). For illustrative purposes, the $\boldsymbol{J} + \boldsymbol{K}$ matrix for the gold dimer Au_2 is shown in Figure 1 distributed among 4 processes according to both the IDD (upper panel) and the BCD (bottom panel) distribution schemes.

The way the local subsets of the global $\boldsymbol{J} + \boldsymbol{K}$ matrix are represented on each of the parallel processes significantly differs in the two distribution schemes. In the BCD scheme, the local subset on one process is represented as a rectangular array whose dimensions are both approximately $2N/P$. (These arrays result from "joining" the black squares of the bottom panels of Figure 1.) We have chosen, in fact, BCD blocks of fixed square size but the blocks at the end of a row or of a column are in general rectangular. Due to the regularity of such distribution scheme, simple formulas (see ref 33) can be used to compute the local indices of the rectangular array from the global matrix indices and vice versa, and also to compute the process ID ("rank", from now on) to which a BCD element belongs from its global indices. In the IDD scheme, instead (see the top panels of Figure 1 for details), the distribution is much less regular with local subsets on each process made up of rectangular blocks of different sizes. A convenient and efficient representation is obtained using a derived data type, composed of a two-dimensional array and some metadata describing its size and placement in the global matrix. On each process, an array of such derived data types is then used to identify each local IDD block.

The two steps (ii, matrix construction, and iii, linear algebra) involving the same matrices in two different distribution schemes were bridged together during the execution by an efficient mapping procedure for exchanging the matrix elements between the parallel processes in order to switch the distribution scheme from IDD (suitable for step ii) to BCD (suitable for step iii). According to such procedure, once the $\boldsymbol{J} + \boldsymbol{K}$ IDD blocks have been computed on the various processes, the program enters a communication phase where each process, in turn, plays the role of the sender while the other processes act as receivers. The sender broadcasts some preliminary information to all of the other processes and sends properly packed data to each of them; the receivers allocate memory buffers according to the broadcast information, receive the data and unpack them according to the BCD scheme. The algorithm for the mapping from a generic "origin distribution" (OD) to a "destination distribution" (DD) can be summarized in 5 steps as follows:

1. **setup** an "info" array of dimension P whose i-th element contains the number of matrix elements to be sent to the i-th process. This step involves, for each of the owned elements, a mapping from the OD local indices to the DD process rank for that element.
2. **allocate** send-buffers (as many as P) according to the "info" array.
3. **pack** the owned data (along with the related destination indices) into the send-buffers. In this phase the matrix elements are packed together according to their DD rank and a derived data type is used to host, along with the very value of the matrix element, also the related DD local indices.
4. **communicate**

```
loop on processes
    if my turn then
        make my 'info' array available to all and send packed data
    else
        allocate receive-buffers according to the 'info' array
        made available by the sender, receive and unpack data,
        deallocate receive-buffers
    end if
end loop
```

   On the receiving processes, the matrix elements are in this phase copied from the buffer into the destination array according to the DD local indices associated with them.
5. **deallocate** send-buffers

This practically led to the removal of the memory allocation associated with the $\boldsymbol{J} + \boldsymbol{K}$ matrix on the single processes when using a sufficiently large number of processors, as both the IDD and the BCD partitioning grain was so fine already for the small Au_2 system of Figure 1 that only a very small amount of memory was required on each parallel process to represent the global matrix. Still, the $\boldsymbol{J} + \boldsymbol{K}$ matrix was only one of the five DKS-sized matrices required. Besides, the density fitting step i, which was left untouched in its serial implementation, turned out to impose severe limitations on the efficiency of the code both in terms of memory and time when an (already moderately) high number P of processes was used. On the time side, due to Amdahl's law,[34] this serial portion of code limited the speedup, and whereas, as already mentioned, the density fitting step was only about 10% of the total SCF iteration time for a serial execution on Au_{16}, it turned out to take a fraction as high as 60% when 64 processors were used and even 75% when 256 processors were used. The major shortcoming of such serial legacy was, however, on the memory side. In fact, the "fat" node responsible for carrying out the density fitting step had to host the whole density and Hermite density matrices (eqs 17 and 18) thus imposing a memory barrier impossible to surmount.

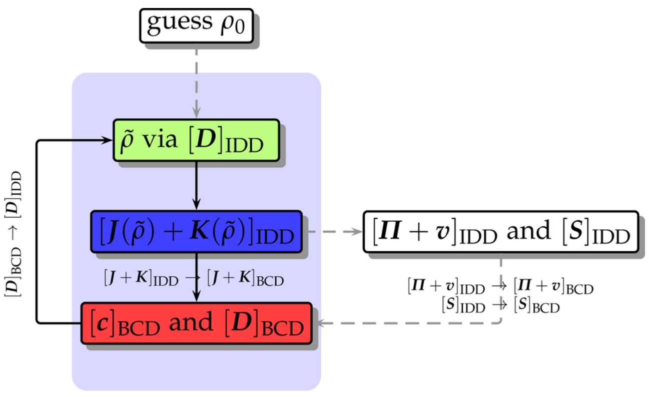
3.2. Full Parallel Implementation. A full parallel implementation of the program is now achieved as a result of (i) the parallelization of the density fitting step and the subsequent removal of any serial portion of code and (ii) the extension and adoption of the distributed matrix approach for all of the five DKS-sized matrices involved in the SCF procedure. As already mentioned, the generalization of the distributed memory approach to the one-electron $\boldsymbol{\Pi} + \boldsymbol{v}$ and overlap $\boldsymbol{S}$ matrices could be carried out with small effort by adapting to their computation the same algorithm as for the $\boldsymbol{J} + \boldsymbol{K}$ matrix. Though not providing any great advantage in terms of time requirements (recall that these two matrices are computed only once in the SCF procedure), the adoption of a distributed matrix approach for the one-electron and overlap matrices made scalable the memory allocation for three out of the five required matrices. A fourth matrix, the eigenvectors $\boldsymbol{c}$ matrix, is computed in a distributed memory fashion by the diagonalization ScaLAPACK routine and is used to construct the density $\boldsymbol{D}$ matrix by another ScaLAPACK routine (again in a distributed memory fashion) so that it is never required entirely on a single process. The output $\boldsymbol{D}$ matrix is computed, according to ScaLAPACK, in the BCD distribution scheme and was, in the previous implementation, collected on the process in charge of carrying out the (serial) density fitting. The permanent distribution of this fifth and last matrix could be obtained by parallelizing the density fitting step as follows.

The first computational kernel to be targeted has been the computation of the Hermite density matrix $\boldsymbol{H}_0$ of eq 18. The Hermite density matrix $\boldsymbol{H}_0$ in BERTHA is represented as a linearized array whose indices map onto the allowed values of α, i, j, and k. A distributed memory parallel computation of $\boldsymbol{H}_0$ was obtained by partitioning the α range into chunks and cyclically assigning each chuck to the various parallel processes. Once the $\boldsymbol{H}_0$ chunks have been computed, the program moves to the evaluation of the vector $\boldsymbol{v}$ (eq 17). This has been parallelized in such a way that each process evaluates the sum in eq 17 over the α ranges that it owns, so that a simple MPI all_reduce/sum operation is required for all ranks to have the correct $\boldsymbol{v}$ vector and go ahead solving the linear equation system to get, each, the array $\boldsymbol{d}$ required for the $\boldsymbol{J} + \boldsymbol{K}$ matrix construction step. The computation of the vector $\boldsymbol{w}$ (eq 19) is somewhat faster than the two computational kernels ($\boldsymbol{H}_0$ and $\boldsymbol{v}$) just discussed; moreover, it does not affect the memory requirements. It has been therefore parallelized in a coarser grain, where the number of parallel tasks equals the number of the atomic centers.

This parallel scheme not only leads to the removal of the remaining serial portions of code but also actually represents a virtually open-ended implementation, permitting the permanent distribution of the density matrix $\boldsymbol{D}$. In fact, the α-range chunks map exactly onto blocks of the density $\boldsymbol{D}$ matrix that are precisely those of the IDD distribution scheme, so that a simple backward mapping (from the BCD to the IDD scheme) is required for the $\boldsymbol{D}$ matrix immediately before the density fitting step (ie., soon after its computation) in order to have it partitioned among the processes in the proper way for the parallel calculation of the Hermite density matrix.

The algorithm for the backward (BCD to IDD) mapping is essentially the same, in reverse, as the forward (IDD to BCD) one already outlined in the previous subsection. The main difference between the two procedures lies in the pack and unpack phases, where a set of destination indices has to be computed as a function of a set of origin indices. As already mentioned, this operation is straightforward in the IDD to BCD mapping due to the regularity of the BCD scheme, whereby local BCD indices can be worked out from the global indices with simple analytic formulas. The same operation is less straightforward in the backward direction, as the IDD decomposition is essentially irregular (with, in addition, only lower triangles being computed, different block sizes, and nonsquare blocks). We use therefore an additional BCD-sized array on each process to host for each i, j local BCD indices the set of four integers determining the related destination indices (rank, block number, local IDD i and j). Note that in our working precision (4 bytes for integer, 8 for real, 16 for complex variables) such an array requires exactly the same amount of memory as the BCD matrix subset. This additional memory is part of the memory allocation that scales linearly with the number of processors.

These improvements notably simplified the flowchart of the DKS module of BERTHA. In particular, with the present implementation, the flowchart is identical for all the parallel processes involved and it perfectly superimposes on the original (serial) SCF procedure. The original SCF procedure is in fact replicated on all the parallel processes with the computational effort required by each of them decreasing with the number of processors used as a consequence of the fact that each process works on subsets of the global matrices. Accordingly, the flowchart of a typical DKS single point run in BERTHA is summarized in Figure 2. After a preliminary "guess ρ_0" phase



guess ρ_0
$\tilde{\rho}$ via $[\boldsymbol{D}]_{\mathrm{IDD}}$
$[\boldsymbol{J}(\tilde{\rho})+\boldsymbol{K}(\tilde{\rho})]_{\mathrm{IDD}}$
$[\boldsymbol{\Pi}+v]_{\mathrm{IDD}}$ and $[S]_{\mathrm{IDD}}$
$[J+K]_{\mathrm{IDD}} \to [J+K]_{\mathrm{BCD}}$
$[\boldsymbol{\Pi}+v]_{\mathrm{IDD}} \to [\boldsymbol{\Pi}+v]_{\mathrm{BCD}}$
$[S]_{\mathrm{IDD}} \to [S]_{\mathrm{BCD}}$
$[c]_{\mathrm{BCD}}$ and $[\boldsymbol{D}]_{\mathrm{BCD}}$
$[\boldsymbol{D}]_{\mathrm{BCD}} \to [\boldsymbol{D}]_{\mathrm{IDD}}$

Figure 2. Flowchart for a single-point DKS calculation in BERTHA. Within the frame are the steps involved in the SCF iterations. The steps outside the frame and linked with gray dashed arrows are executed only once. The workflow is identically replicated on all the parallel processes, each process working on subsets (square-bracketed quantities) of the global matrices either in the IDD or in the BCD distribution scheme. The mapping from one scheme to another is also indicated across the flow arrows between two steps of the execution.

(providing a guess density to the SCF cycle and for which, though it is not detailed here, an analogous distributed memory parallel approach has also been implemented) the SCF cycle is entered with its (i) density fitting step, (ii) matrix construction step, and (iii) linear algebra step. The total explicit communications (besides those internal to the ScaLAPACK routines) are the mapping of the $\boldsymbol{J}$ + $\boldsymbol{K}$ matrix from IDD to BCD (also of the $\boldsymbol{\Pi}$ + $\boldsymbol{v}$ and $\boldsymbol{S}$ matrices at the first iteration), the mapping of the $\boldsymbol{D}$ matrix from BCD to IDD and the reduction of $\boldsymbol{v}$ and $\boldsymbol{w}$.

The linear algebra phase is 3-fold: the "level shift" phase, in which the DKS matrix needs to be transformed from basis function space to spinor space, the diagonalization phase that solves the generalized eigenvalue problem for the Hermitian matrix $\boldsymbol{H}_{\mathrm{DKS}}$ (eq 5), and the matrix multiplication phase of eq 12 to obtain the density matrix from the occupied positive-energy spinors.

4. PERFORMANCE

To evaluate the performance of the above outlined implementation, we carried out a first batch of test calculations on a representative set of gold clusters: Au_8, Au_{16}, Au_{32} (the related geometries are given as Supporting Information (SI)). The corresponding DKS matrix sizes are 6304 for Au_8 (0.59 GiB), 12608 for Au_{16} (2.37 GiB), 25216 for Au_{32} (9.47 GiB). For each gold atom the large component of the G-spinor basis set was derived by decontracting the double-ζ quality basis set of Dyall[49,50] (22s 18p 13d 8f). The corresponding small-component basis was generated using the restricted kinetic balance relation.[51] The auxiliary HGTF basis set optimized by us and denoted as B16[20] was used for the density fitting, resulting in 210 auxiliary functions for each Au atom (the total number N_{aux} of auxiliary functions is 1680, 3360, and 6720 for Au_8, Au_{16}, and Au_{32}, respectively). A numerical integration grid of 24 566 points for each gold atom was employed.

In order to assess the load balance in less "symmetric" cases, calculations were repeated on a molecule containing atoms of varied size. We chose the organometallic compound $[(Ph_3P)Au(C_2H_2)]^+$ (see SI for the related geometry) with a DKS matrix size ($2N$ = 6072) comparable to the Au_8 case. For this system, a decontracted triple-ζ quality basis set (30s 24p 15d 11f 1g) of Dyall[49,50] for the large component of the G-spinor basis coupled to the auxiliary HGTF optimized B20[20] basis set was used for the gold atom. Large component basis functions for P, C, and H were derived by decontracting the related Def2-TZVPP[52] basis sets available at the "Basis Set Exchange" site.[53] The corresponding small-component basis was generated using the restricted kinetic balance relation.[51] For each P, C, and H atom, an auxiliary HGTF basis set was automatically generated from the related orbital basis set following the procedure devised in ref 54 using the "size-control parameters" l_{MAXINC} = 1 and f_{sam} = 1.5. This led to a total number N_{aux} of auxiliary functions of 2976. A numerical integration grid of 24566, 7066, 6022, and 3686 was employed for each Au, P, C, and H atom, respectively.

For all calculations, the exchange-correlation functional adopted is the BLYP functional made of the Becke 1988 exchange (B88)[55] plus the Lee−Yang−Parr (LYP) functional.[56]

Calculations were performed on a Beowulf cluster (mccw from now on) equipped with Intel(R) Xeon(R) CPU E5-2670 0 2.60 GHz (24 nodes, 384 cores with 128 GiB/node, 8 GiB/core) and Infiniband network. Preliminary tests to assess optimal ScaLAPACK parameters confirmed on mccw what we had previously found for an other architecture in ref 30: the ScaLAPACK block size adopted was accordingly, 32 square process grids (P_r = P_c) were preferred to rectangular process grids, and rectangular process grids with P_r < P_c were chosen, rather than those with P_r > P_c.

The test calculations were addressed to the following issues:

Table 1. Wall-Clock Time in Seconds (Average over 4 SCF Cycles) Spent in the Distribution Mapping Routines during Calculations on Au_8, Au_{16}, and Au_{32}[a]

	$[(Ph_3P)Au(C_2H_2)]^+$		Au_8		Au_{16}		Au_{32}	
P	$t_{BCD\to IDD}$	$t_{IDD\to BCD}$	$t_{BCD\to IDD}$	$t_{IDD\to BCD}$	$t_{BCD\to IDD}$	$t_{IDD\to BCD}$	$t_{BCD\to IDD}$	$t_{IDD\to BCD}$
4	0.70	0.57	0.74	0.60	3.52	2.46	20.61	9.68
8	0.39	0.36	0.41	0.38	1.85	1.46	9.09	6.22
16	0.21	0.25	0.22	0.24	0.96	0.98	5.96	3.72
32	0.20	0.24	0.21	0.25	0.73	0.87	2.95	3.32
64	0.35	0.38	0.36	0.40	0.82	1.00	2.55	3.27
128	(2.5%) 0.92	(2.3%) 0.86	(2.9%) 0.90	(2.8%) 0.88	1.50	1.64	3.23	4.02
256	(6.9%) 2.69	(6.6%) 2.56	(6.5%) 1.84	(7.4%) 2.11	(2.6%) 3.84	(2.4%) 3.64	6.10	6.69

[a]Percentages of the SCF iteration times are smaller than 1% in any case except where otherwise noted in parentheses.

Table 2. Memory Per Process Peak (Average Value M_{av}, Maximum Positive Δ_+ and Negative Δ_- Deviations) in MiB over P Processes for Calculations on $[(Ph_3P)Au(C_2H_2)]^+$, Au_8, Au_{16}, and Au_{32} Using a Number of Processes P from 4 to 256

	$[(Ph_3P)Au(C_2H_2)]^+$			Au_8			Au_{16}			Au_{32}		
P	Δ_-	M_{av}	Δ_+	Δ_-	M_{av}	Δ_+	Δ_-	M_{av}	Δ_+	Δ_-	M_{av}	Δ_+
4	76	1842	37	34	1882	60	81	6214	46	126	23835	187
8	57	1253	59	64	1303	71	54	3770	90	88	14106	232
16	12	884	15	8	888	44	14	2124	23	8	7405	63
32	62	799	84	33	773	96	44	1499	85	53	4827	155
64	47	700	92	37	672	92	80	1167	80	64	2880	83
128	21	631	115	24	620	110	54	961	81	48	2220	101
256	10	599	134	15	586	119	52	866	83	80	1942	92

1. Is the distribution mapping algorithm (both forward and backward) efficient?
2. Are matrices equally distributed among the processes (is load balance achieved)?
3. How does the memory used by each process vary throughout the various steps of an SCF iteration?
4. How does the memory used by each process scale with the number of computing cores?
5. How does the execution time scale with the number of computing cores?
6. Portability and numerical stability of the code.

All calculations were performed using a number of computing cores equal to the number P of parallel processes. Memory measurements were carried out on mccw by an external script extracting the value (VMSIZE) of the virtual memory usage from the/proc/$pid/status file for each parallel process (identified by the ID number $pid) every ten seconds approximately. It is worth stressing here that such value represents the total amount of memory of a process, including its resident size, swap size, code, data, shared libraries, and all MPI buffers. Except where otherwise noted, a maximum number of SCF iterations equal to 5 was set.

1. The wall-clock execution time in seconds (average over 4 SCF cycles) spent in the distribution mapping routines (both from BCD to IDD and from IDD to BCD) is reported in Table 1 for $[(Ph_3P)Au(C_2H_2)]^+$, Au_8, Au_{16}, and Au_{32} using a number P of processes from 4 to 256. With the exception of the very top right edge of the table (where huge data sets are to be exchanged among few processes), in all cases, the mapping is performed in few seconds, and except where noted (bottom left corner of the table, where small data sets are to be exchanged among many processes), it represents less than 1% of the total SCF iteration time. The increasing trend along a row of the table, that is, with increasing system size, is clearly due to the increasing amount of data to be exchanged. The trend along a column, that is, with increasing P, is instead less straightforward: with P going from 4 to 256, the time spent in the mapping routines decreases down to a minimum and then tends to increase again. This is likely the result of two opposite trends: smaller send-buffers with increasing P require less communication times but higher is the overhead due to the loop in the "communicate" phase of step 4 of the algorithm (see section 3).

2. Figure 1 (section 3) should give a clear idea of the balance of the data- and work-load among the processes due to the matrix distribution both in the IDD and the BCD scheme. The grain, in fact, is already so fine for the small Au_2 system that the matrices appear equally distributed among the four processes considered there. Quantitative information can however be obtained checking the memory peak reached by each process during their execution. Table 2 reports the average (M_{av}) value of the memory peak in MiB over P processes (along with its maximum positive Δ_+ and negative Δ_- deviations) for the four testcases using a number of processes P from 4 to 256. It is worth stressing here that the values in Table 2 have been taken throughout the whole execution of BERTHA, including the preliminary "guess ρ_0" step of Figure 2 and the first special iteration involving the construction of the one-electron and overlap matrices, so that they truly represent the minimum, average, and maximum height of the memory barrier for the respective calculation. Throughout the table, the gap $\Delta_+ + \Delta_-$ is about 150 MiB on average. It reaches its maximum of 320 MiB for Au_{32}, $P = 8$, which, however, is only 2% of the corresponding M_{av}. The worst ratio $(\Delta_+ + \Delta_-)/M_{av}$ is seen for the $P = 256$ runs, being 24% for $[(Ph_3P)Au(C_2H_2)]^+$, 23% for Au_8, 16% for Au_{16}, and 9% for Au_{32}. The code proves, therefore, to be largely balanced in the usage of memory, with the balance improving with increasing system size. We can now analyze, therefore, the detailed profile of the memory usage per process during execution.

3. The virtual memory used by process 0 (largely representative of the other processes as just discussed) for a complete SCF iteration on the Au_{16} cluster using $P = 4$ is shown in Figure 3. The time zero is set to the start of the "density" phase of the previous SCF cycle and it is from this phase that we start the analysis because at this point the memory allocation is at its base value. Such base value corresponds to the allocation of the BCD subsets of the five DKS-sized matrices required, plus an additional array containing the BCD to IDD mapping information for each of the local BCD indices, plus a "residual" amount of memory that can be split into a "default" contribution (due to automatic allocation and to the initialization of the parallel environment) and a system dependent contribution mainly related to the density fitting step. (This explains why in Table 2 M_{av} for $[(Ph_3P)Au(C_2H_2)]^+$ overtakes M_{av} for Au_8 at high values of P: as discussed in the input details at the head of this section, the spinor basis is smaller for $[(Ph_3P)Au(C_2H_2)]^+$ while the auxiliary basis is larger.) This residual amount of memory depends little on the parallel structure (variations are within one hundred MiB, increasing with the number of processes) but has a more pronounced dependence on system size, reaching its maximum of 499, 564, and 822 MiB for Au_8, Au_{16}, and Au_{32}, respectively.

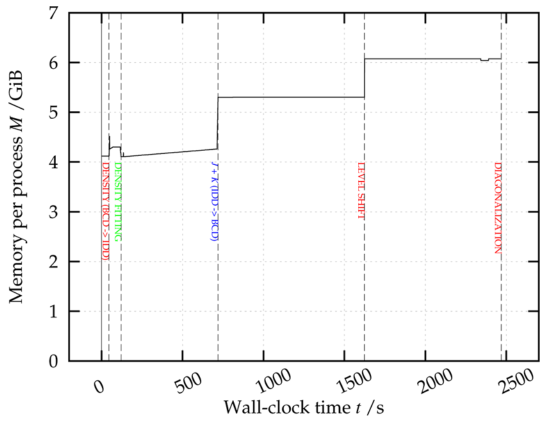


Figure 3. Virtual memory used by a single process (process 0) as a function of the execution time for one SCF cycle of a calculation on Au_{16} using four processes.

Once the $\boldsymbol{D}$ matrix is computed in the BCD scheme in the "density" phase from the eigenvectors computed in a previous cycle, it is mapped to the IDD scheme for use in the "density fitting" step. Additional memory (see the corresponding step in the plot) is here required to host the IDD blocks of $\boldsymbol{D}$. The parallel density fitting is then carried out and, soon after, the program enters the matrix construction ($J + K$ in the plot) step where the IDD blocks of $\boldsymbol{J} + \boldsymbol{K}$ are successively allocated (see the increasing trend) and computed. At this point, the $\boldsymbol{J} + \boldsymbol{K}$ matrix is mapped from the IDD to the BCD scheme and the two costly linear algebra substeps "level shift" and diagonalization can be carried out. The memory steps in these two phases are due to the replication of two of the DKS-sized matrices in the "level shift" phase and to the allocation of additional workspace in the diagonalization phase.

4. Having outlined the memory usage profile throughout the various SCF computational kernels, we have now to analyze how this memory profile scales with the number P of processes. As just discussed, the "bulk" memory allocation is due to five BCD-sized matrices plus a mapping array, the rest being related to automatic allocation, parallel environment, and additional working space. For greater clarity, in the following, we will discuss the scaling of the overall memory allocated by the single processes. The left panel of Figure 4 reports the memory used by process 0 (again, this is essentially the same as any other process) as a function of the wall-clock execution time during 4 SCF iterations on Au_{16}. Seven curves (six of them are filled for a better visualization) are displayed, one for each run employing a different number P of processes which, from top to bottom, is 4, 8, 16, 32, 64, 128, and 256. (Thus, one of the four periodic portions of the upper ($P = 4$) curve is exactly the same curve as in Figure 3.) The final point of each curve (indicating the memory used at the end of the corresponding run, which we have seen to be maximal) is connected by an

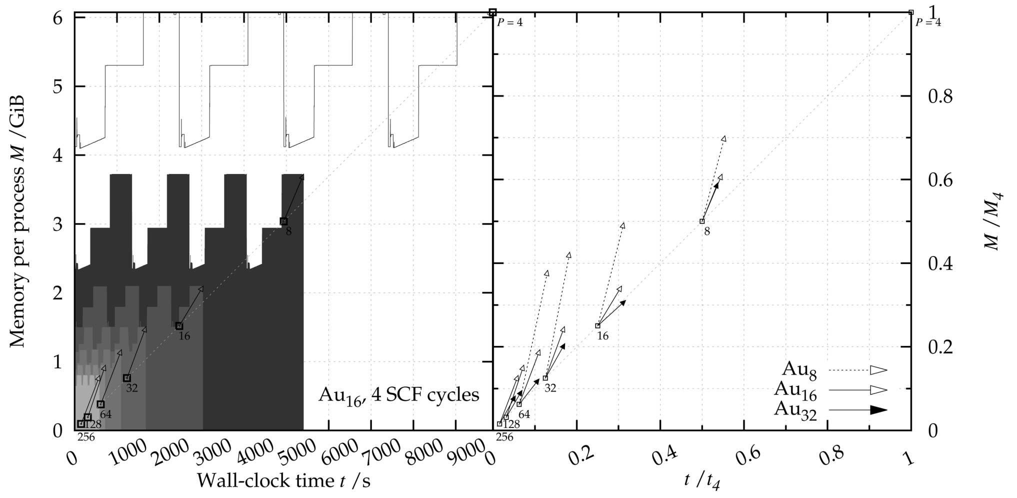


Figure 4. Left: virtual memory used by one process (process 0) as a function of the execution time during 4 SCF cycles of a calculation on Au_{16}, using a number of processors P increasing in doubling steps from 4 (upper curve) to 256 (bottom curve). Arrows are displayed joining the ideal and the actual performance points relative to the $P = 4$ case (see the text for a discussion). Right: same plot including Au_8 (up to $P = 64$) and Au_{32}, and using scaled coordinates and retaining only the arrows. The reference values (t_4, M_4) are (1295 s, 1.83 GiB), (9874 s, 6.07 GiB) and (77849 s, 23.3 GiB) for Au_8, Au_{16}, and Au_{32}, respectively.

arrow to a point on the diagonal of the plot that represents, relative to the $P = 4$ case, the ideal (time, memory) linear performance, which halves at each P doubling step. Thus, the length of each arrow is a measure of the efficiency of the corresponding run, both in terms of memory and time requirements. The time scaling will be discussed later on. While the arrows appear to indicate that the memory scaling is somewhat worse than the time scaling (the y component of the arrows is longer than the x component), this is an effect of the included residual (nonscalable) workspace, which is very small in absolute practical terms (few GiB at most) even on massively parallel systems and grows very little with system size.

The right panel of Figure 4 shows the same arrows as the left panel using scaled x and y units, which permits the inclusion of the data relative also to the Au_{32} and Au_8 systems (only results up to $P = 64$ have been reported in this last case, as higher P runs turned out to be too fast to measure the memory usage with our script). For Au_8, Au_{16}, and Au_{32} the reference values (t_4, M_4) are (1295 s, 1.83 GiB), (9874 s, 6.07 GiB) and (77849 s, 23.3 GiB), respectively. As the figure makes clear, the arrows tend to collapse in going from Au_8 (dashed line empty head) to Au_{16} (solid line empty head) to Au_{32} (solid line filled head) and the memory usage per process is kept within 2 GiB using 256 processors even for the costly Au_{32} case.

5. The speedup (relative to $P = 4$) for all of the computational kernels involved in the SCF procedure is reported in Figure 5 for the Au_{32} case as a function of the number P of processes used. The plot shows excellent scaling in the considered range of P (4−256) for many of the routines including the costly $\boldsymbol{J} + \boldsymbol{K}$ construction step (blue line) with the notable exception of two of them: the "w" and the diagonalization phases. However, the "w" phase (ie., the parallel computation of the vector $\boldsymbol{w}$ that, as mentioned in section 3, was parallelized in a coarse grain approach) represents 1.7% of the total SCF iteration in the worst case ($P = 256$) and therefore practically does not affect the overall performance. Figure 5 shows therefore that the only limitation to the overall SCF speedup is due to the matrix diagonalization step. Diagonalization free approaches, in this respect, such as the Thouless expansion-based one[57] that has recently been extended to the four-component formalism,[58] present themselves as a promising alternative.

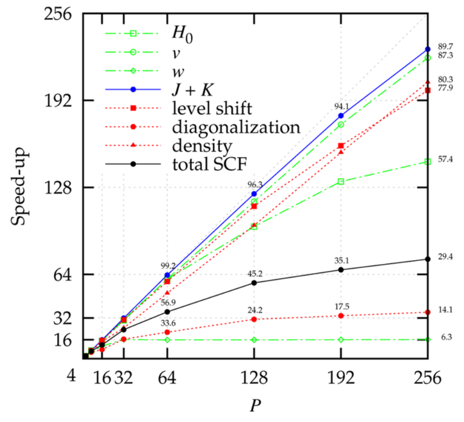


Figure 5. Speedup for all of the computational kernels of the SCF procedure for Au_{32} as a function of the number of processors P. Linear scaling is traced with a dotted diagonal line. Efficiency (in percentage) is reported for all kernels at $P = 256$, and for the matrix construction ('$J + K$') step, the diagonalization step and the total SCF iteration also at $P = 64$, 128, and 192.

6. Over the years, we ported BERTHA on several parallel computers featuring different architectures: a Beowulf-type workstation,[25] an SGI Altix 4700,[30] an IBM SP Power 6[31] and, with the present work, also on the BlueGene/Q Fermi (FERMI from now on) located at CINECA, Italy and equipped with IBM PowerA2 1.6 GHz (10240 nodes, 163840 cores with 16 GiB/node, 1GiB/core) and a 11 links → 5D Torus network interface.

We report in Figure 6 a comparison of the speedup of the $\boldsymbol{J} + \boldsymbol{K}$ matrix construction including the IDD to BCD mapping on FERMI and on mccw for the Au_{16} gold cluster. (We could not report tests on FERMI for the Au_{32} case as the computing time allocation was over when we finalized the current distributed memory implementation. It should be noted that, with the previous versions, we could not carry out calculations on Au_{32} because of the limited amount of memory per node available on FERMI.) The speedup limit is about $P = 128$ on mccw. Although the speedup limit is much beyond this value on FERMI, we should note that absolute execution times are much slower (by two to three times depending on the specific routines) because of the lower FERMI CPU frequency. In general, we conclude that the FERMI machine is not suitable for the present version of BERTHA and for the problems of the size that we currently manage to handle.

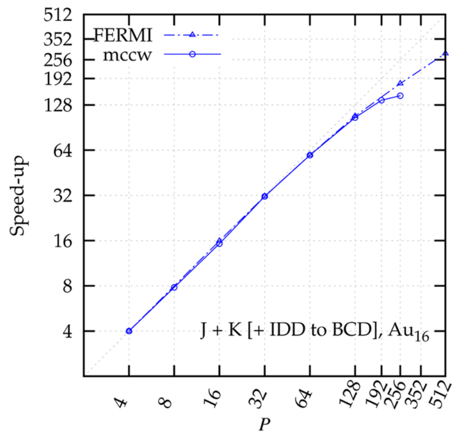


Figure 6. Speedup for the $\boldsymbol{J} + \boldsymbol{K}$ matrix construction step plus IDD to BCD mapping for Au_{16} as a function of the number of processes P on mccw (solid line) and on FERMI (dashed-dotted line). Linear scaling is indicated by a dotted diagonal line. Note that a logarithmic scale is employed for both axes.

It is important to mention that the substantial changes introduced in the present full parallel implementation do not affect at all the numerical stability of the code. Convergence on the total energy within 10^{-9} Hartree was achieved for Au_8 and Au_{16}, and within 10^{-8} Hartree for the organometallic compound $[(Ph_3P)Au(C_2H_2)]^+$ using 128 processes. The maximum element of the density matrix difference between the last two cycles is 10^{-6} for the gold clusters and 10^{-5} for the organometallic compound.

As a sample application, we finally tested the effective usability of the code on a simple perovskite model $[Pb_4I_{20}]^{12-}$ (see SI for related geometry) made up of four $[PbI_6]^{4-}$ octahedra as shown in Figure 7 using a double-ζ (DZ) and a triple-ζ (TZ) quality basis set.

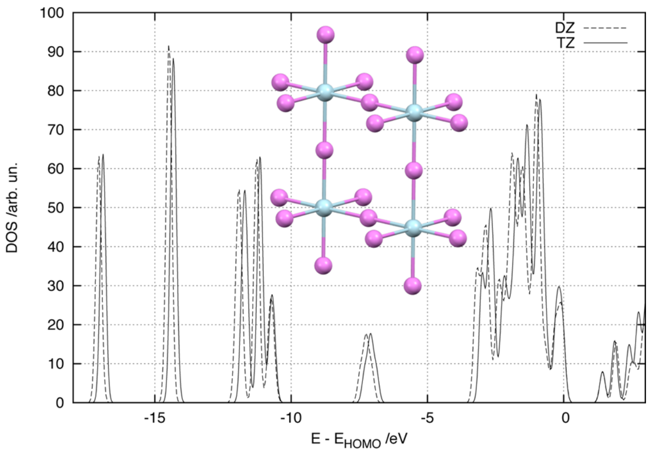


Figure 7. Density of states for the $[Pb_4I_{20}]^{12-}$ perovskite model (the studied structure is superimposed to the plotting area) computed using a double-ζ (DZ, dashed line) and a triple-ζ (TZ, solid line) quality basis set and a Gaussian broadening of 0.1 eV. The value of the HOMO energy has been taken as energy zero in both cases.

For the DZ calculation, the large component basis set was derived by decontracting the double-ζ quality basis set of Dyall[59,60] resulting in (24*s* 20*p* 14*d* 9*f*) functions for Pb and (24*s* 20*p* 14*d* 9*f*) functions for I. The corresponding small-component basis was generated using the restricted kinetic balance relation.[51] The DKS matrix size is 13712 (2.80 GiB). A set of 3920 auxiliary functions was generated from the orbital basis set using the same procedure as for the gold clusters and relaxing the f_{sam} parameter (see ref 54) in order to obtain 18 exponents.

For the TZ calculation, the large component basis set was derived by decontracting the triple-ζ quality basis set of Dyall[60−62] resulting in (30*s* 16*p* 17*d* 11*f*) functions for Pb and (28*s* 21*p* 15*d* 1*f*) functions for I. The corresponding small-component basis was generated using the restricted kinetic balance relation.[51] The DKS matrix size is 18160 (4.91 GiB). A set of 4952 auxiliary functions was generated from the orbital basis set using the same procedure as for the gold clusters and relaxing the f_{sam} parameter (see ref 54) in order to obtain 22 exponents. For both the DZ and TZ calculations, the B88P86 functional[55,63,64] was used.

Convergence up to 3×10^{-6} Hartree on the absolute energy was reached for the DZ and the TZ case. Convergence within 4 meV (<0.015 mHartree) on the band gap using the two basis sets (DZ: 1.424 eV. TZ: 1.428 eV) was reached. The density of states (DOS) shifted on the HOMO energy in the (−18, 3) eV range is shown for the DZ basis (dashed line) and TZ basis (solid line) in Figure 7.

5. CONCLUSIONS

The present work is the conclusion of a series of efforts to overcome both the time and the memory bottlenecks that have so far restricted the applicability of all-electron four-component relativistic DFT approaches to the modeling of the electronic structure of chemical systems containing heavy elements. We have now completed a full parallel implementation of the DKS module of the program BERTHA featuring (i) no serial portions of code and (ii) a complete distributed memory approach. At its core is an efficient procedure for switching between two matrix distribution schemes, one optimal for the parallel construction of the matrices, guided by the structure of our specific basis sets and by the density fitting algorithms used, and another required by the ScaLAPACK routines employed to perform linear algebra operations. Extensive testing of the new code shows that only the complex matrix diagonalization step (at present executed using the ScaLAPACK library) hinders somewhat efficient time scaling, while the other time-consuming phases of the DKS calculation scale nearly ideally and the memory bottleneck associated with potentially huge matrices is entirely removed. Indeed, all-electron four-component DKS calculations on systems as costly as the Au_{32} gold cluster, with more than 25 000 basis functions are now feasible with BERTHA provided that a minimal amount of memory per core (as small as 2 GiB) is available. It is particularly relevant that the linear scaling achieved for the density fitting and integral evaluation steps opens the way for an efficient implementation of the exact exchange.[16] This is precisely what we are working on at the moment.

■ ASSOCIATED CONTENT

Ⓢ Supporting Information

Geometries (XYZ format) of the systems used for the test calculations: namely, the Au_8 (SI file 1), Au_{16} (SI file 2), and Au_{32} (SI file 3) gold clusters, the organometallic compound $[(Ph_3P)Au(C_2H_2)]^+$ (SI file 4) and the perovskite model $[Pb_4I_{20}]^{12-}$ (SI file 5). This material is available free of charge via the Internet at http://pubs.acs.org.

■ AUTHOR INFORMATION

Corresponding Authors

*Email: srampino@thch.unipg.it.
*Email: belp@thch.unipg.it.
*Email: loriano@storchi.org.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This work has been supported by the Italian MIUR through the FIRB-futuro-in-ricerca project RBFR1022UQ. The authors thank the Italian SuperComputing Resource Allocation (ISCRA) through the project HP10BT1TD8 (BERTHA) and CNR-ISTM for the allocation of computational resources.

■ REFERENCES

(1) Pyykkö, P.; Desclaux, J. P. *Acc. Chem. Res.* **1979**, *12*, 276−281.
(2) Grant, I. P. *Relativistic Quantum Theory of Atoms and Molecules: Theory and Computation*; Springer Series on Atomic, Optical, and Plasma Physics; Springer: Berlin, 2007; pp 1−797.
(3) *Relativistic Electronic Structure Theory*; Schwerdtfeger, P., Ed.; Elsevier: Amsterdam, 2002; Vol. *1*, pp 1−946.
(4) Autschbach, J. *J. Chem. Phys.* **2012**, *136*, 150902.
(5) Dirac, P. A. M. *Proc. R. Soc. London, Ser. A* **1928**, *117*, 610−624.
(6) Vignale, G.; Kohn, W. *Phys. Rev. Lett.* **1996**, *77*, 2037−2040.
(7) Kümmel, S.; Kronik, L. *Rev. Mod. Phys.* **2008**, *80*, 3−60.
(8) MacDonald, A. H.; Vosko, S. H. *J. Phys. C: Solid State Phys.* **1979**, *12*, 2977−2990.
(9) Liu, W. *Mol. Phys.* **2010**, *108*, 1679−1706.
(10) Saue, B. T.; Faegri, K.; Helgaker, T.; Gropen, O. *Mol. Phys.* **1997**, *91*, 937−950.
(11) Yanai, T.; Iikura, H.; Nakajima, T.; Ishikawa, Y.; Hirao, K. *J. Chem. Phys.* **2001**, *115*, 8267−8273.
(12) Saue, T.; Helgaker, T. *J. Comput. Chem.* **2002**, *23*, 814−823.
(13) Varga, S.; Engel, E.; Sepp, W.-D.; Fricke, B. *Phys. Rev. A* **1999**, *59*, 4288−4294.

(14) Liu, W.; Hong, G.; Dai, D.; Li, L.; Dolg, M. *Theor. Chem. Acc.* **1997**, *96*, 75−83.

(15) Shiozaki, T. *J. Chem. Theory. Comput.* **2013**, *9*, 4300−4303.

(16) Kelley, M. S.; Shiozaki, T. *J. Chem. Phys.* **2013**, *138*, 204113.

(17) Belpassi, L.; Storchi, L.; Quiney, H. M.; Tarantelli, F. *Phys. Chem. Chem. Phys.* **2011**, *13*, 12368−12394.

(18) Vahtras, O.; Almlöf, J.; Feyereisen, M. W. *Chem. Phys. Lett.* **1993**, *213*, 514−518.

(19) Eichkorn, K.; Treutler, O.; Öhm, H.; Häser, M.; Ahlrichs, R. *Chem. Phys. Lett.* **1995**, *240*, 283−290.

(20) Belpassi, L.; Tarantelli, F.; Sgamellotti, A.; Quiney, H. M. *J. Chem. Phys.* **2006**, *124*, 124104.

(21) Belpassi, L.; Tarantelli, F.; Sgamellotti, A.; Quiney, H. M. *Phys. Rev. B* **2008**, *77*, 233403.

(22) Iliaš, M.; Saue, T. *J. Chem. Phys.* **2007**, *126*, 064102.

(23) Liu, W.; Peng, D. *J. Chem. Phys.* **2006**, *125*, 044102.

(24) Peng, D.; Liu, W.; Xiao, Y.; Cheng, L. *J. Chem. Phys.* **2007**, *127*, 104106.

(25) Belpassi, L.; Storchi, L.; Tarantelli, F.; Sgamellotti, A.; Quiney, H. M. *Fut. Gener.Comput. Syst.* **2004**, *20*, 739−747.

(26) NVIDIA, CUDA Programming Guide 2.0. http://docs.nvidia.com/cuda/ (accessed February 20, 2014).

(27) Nadathur, S. *Fast Sort on CPUs, GPUs and Intel MIC Architectures*; Technical Report, Intel Labs, 2010.

(28) Adiga, N. R.; Almási, G.; Aridor, Y.; Barik, R.; Beece, D. K.; Bellofatto, R.; Bhanot, G.; Bickford, R.; Blumrich, M. A.; Bright, A. A.; Brunheroto, J. R.; Cascaval, C.; Castaños, J. G.; Chan, W.; Ceze, L.; Coteus, P.; Chatterjee, S.; Chen, D.; Chiu, G. L.-T.; Cipolla, T. M.; Crumley, P.; Desai, K. M.; Deutsch, A.; Domany, T.; Dombrowa, M. B.; Donath, W. E.; Eleftheriou, M.; Erway, C. C.; Esch, J.; Fitch, B. G.; Gagliano, J.; Gara, A.; Garg, R.; Germain, R. S.; Giampapa, M.; Gopalsamy, B.; Gunnels, J. A.; Gupta, M.; Gustavson, F. G.; Hall, S.; Haring, R. A.; Heidel, D. F.; Heidelberger, P.; Herger, L.; Hoenicke, D.; Jackson, R. D.; Jamal-Eddine, T.; Kopcsay, G. V.; Krevat, E.; Kurhekar, M. P.; Lanzetta, A. P.; Lieber, D.; Liu, L. K.; Lu, M.; Mendell, M. P.; Misra, A.; Moatti, Y.; Mok, L. S.; Moreira, J. E.; Nathanson, B. J.; Newton, M.; Ohmacht, M.; Oliner, A. J.; Pandit, V.; Pudota, R. B.; Rand, R. A.; Regan, R. D.; Rubin, B.; Ruehli, A. E.; Rus, S.; Sahoo, R. K.; Sanomiya, A.; Schenfeld, E.; Sharma, M.; Shmueli, E.; Singh, S.; Song, P.; Srinivasan, V.; Steinmacher-Burow, B. D.; Strauss, K.; Surovic, C. W.; Swetz, R. A.; Takken, T.; Tremaine, R. B.; Tsao, M.; Umamaheshwaran, A. R.; Verma, P.; Vranas, P.; Ward, T. J. C.; Wazlowski, M. E.; Barrett, W.; Engel, C.; Drehmel, B.; Hilgart, B.; Hill, D.; Kasemkhani, F.; Krolak, D. J.; Li, C. T.; Liebsch, T. A.; Marcella, J. A.; Muff, A.; Okomo, A.; Rouse, M.; Schram, A.; Tubbs, M.; Ulsh, G.; Wait, C. D.; Wittrup, J.; Bae, M.; Dockser, K. A.; Kissel, L.; Seager, M. K.; Vetter, J. S.; Yates, K. An Overview of the BlueGene/L Supercomputer. *Supercomputing, ACM/IEEE 2002 Conference*; IEEE: New York, 2002; pp 1−22, DOI: 10.1109/SC.2002.10017.

(29) Orlando, R.; Delle Piane, M.; Bush, I. J.; Ugliengo, P.; Ferrabone, M.; Dovesi, R. *J. Comput. Chem.* **2012**, *33*, 2276−2284.

(30) Storchi, L.; Belpassi, L.; Tarantelli, F.; Sgamellotti, A.; Quiney, H. M. *J. Chem. Theory. Comput.* **2010**, *6*, 384−394.

(31) Storchi, L.; Rampino, S.; Belpassi, L.; Tarantelli, F.; Quiney, H. M. *J. Chem. Theory. Comput.* **2013**, *9*, 5356−5364.

(32) *Message Passing Interface Forum. MPI: A Message- Passing Interface Standard. Version 2.2*; University of Tennessee: Knoxville, 2009; http://www.mpi-forum.org/ (accessed March 24, 2014).

(33) Blackford, L. S.; Choi, J.; Cleary, A.; D'Azeuedo, E.; Demmel, J.; Dhillon, I.; Hammarling, S.; Henry, G.; Petitet, A.; Stanley, K.; Walker, D.; Whaley, R. C. In *ScaLAPACK User's Guide*; Dongarra, J. J., Ed.; Society for Industrial and Applied Mathematics: Philadelphia, 1997; pp 1−319.

(34) Amdahl, G. M. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *Proceedings of the April 18−20, 1967, Spring Joint Computer Conference, New York*, 1967; pp 483−485.

(35) (a) te Velde, G.; Bickelhaupt, F. M.; Baerends, E. J.; Fonseca Guerra, C.; van Gisbergen, S. J. A.; Snijders, J. G.; Ziegler, T. *J. Comput. Chem.* **2001**, *22*, 931−967. (b) Fonseca Guerra, C.; Snijders, J. G.; te Velde, G.; Baerends, E. J. *Theor. Chem. Acc.* **1998**, *99*, 391−403. (c) ADF2013, SCM, Theoretical Chemistry, Vrije Universiteit, Amsterdam, The Netherlands,http://www.scm.com (accessed May 12, 2014).

(36) (a) Nieplocha, J.; Palmer, B.; Tipparaju, V.; Krishnan, M.; Trease, H.; Apr, E. *Int. J. High Perform. Comput. Appl.* **2006**, *20*, 203−231. (b) Global Arrays Webpage. http://hpc.pnl.gov/globalarrays/ (accessed May 12, 2014).

(37) (a) Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S.; Windus, T. L.; Dupuis, M.; Montgomery, J. A. *J. Comput. Chem.* **1993**, *14*, 1347−1363. (b) Gordon, M. S.; Schmidt, M. W. In *Theory and Applications of Computational Chemistry: the first forty years*; Dykstra, C. E., Frenking, G., Kim, K. S., Scuseria, G. E., Eds.; Elsevier: Amsterdam, 2005; pp 1167−1189.

(38) Fletcher, G. D.; Schmidt, M. W.; Bode, B. M.; Gordon, M. S. *Comput. Phys. Commun.* **2000**, *128*, 190−200.

(39) Quiney, H. M.; Skaane, H.; Grant, I. P. *J. Phys. B: At., Mol. Opt. Phys.* **1997**, *30*, L829.

(40) Quiney, H. M.; Skaane, H.; Grant, I. P. In *Ab Initio Relativistic Quantum Chemistry: Four-Components Good, Two-Components Bad!*; Löwdin, P.-O., Ed.; Advances in Quantum Chemistry; Academic Press: New York, 1998; Vol. 32, pp 1−49.

(41) Quiney, H. M.; Belanzoni, P. *J. Chem. Phys.* **2002**, *117*, 5550−5563.

(42) Belpassi, L.; Tarantelli, F.; Sgamellotti, A.; Quiney, H. M. *J. Chem. Phys.* **2005**, *122*, 184109.

(43) Belpassi, L.; Tarantelli, F.; Sgamellotti, A.; Quiney, H. M. *J. Chem. Phys.* **2008**, *128*, 124108.

(44) Rajagopal, A. K.; Callaway, J. *Phys. Rev. B* **1973**, *7*, 1912−1919.

(45) Challacombe, M.; Schwegler, E.; Almlöf, J. *J. Chem. Phys.* **1996**, *104*, 4685−4698.

(46) Ahmadi, G. R.; Almlöf, J. *Chem. Phys. Lett.* **1995**, *246*, 364−370.

(47) Grant, I. P.; Quiney, H. M. *Int. J. Quantum Chem.* **2000**, *80*, 283−297.

(48) Becke, A. D. *J. Chem. Phys.* **1988**, *88*, 2547−2553.

(49) Dyall, K. G. *Theor. Chem. Acc.* **2004**, *112*, 403−409 Basis sets are available from the Dirac web site, http://dirac.chem.sdu.dk (accessed July 16, 2014).

(50) Dyall, K. G.; Gomes, A. S. *Theor. Chem. Acc.* **2010**, *125*, 97−100 Basis sets are available from the Dirac web site, http://dirac.chem.sdu.dk (accessed July 16, 2014).

(51) Grant, I. P.; Quiney, H. M. *Phys. Rev. A* **2000**, *62*, 022508.

(52) Weigend, F.; Ahlrichs, R. *Phys. Chem. Chem. Phys.* **2005**, *7*, 3297−3305.

(53) Schuchardt, K. L.; Didier, B. T.; Elsethagen, T.; Sun, L.; Gurumoorthi, V.; Chase, J.; Li, J.; Windus, T. L. *J. Chem. Inf. Model.* **2007**, *47*, 1045−1052, https://bse.pnl.gov/bse/portal, (accessed July 16, 2014).

(54) Yang, R.; Rendell, A. P.; Frisch, M. J. *J. Chem. Phys.* **2007**, *127*, 074102.

(55) Becke, A. D. *Phys. Rev. A* **1988**, *38*, 3098−3100.

(56) Lee, C.; Yang, W.; Parr, R. G. *Phys. Rev. B* **1988**, *37*, 785−789.

(57) Noga, J.; Šimunek, J. *J. Chem. Theory. Comput.* **2010**, *6*, 2706−2713.

(58) Hrdá, M.; Kulich, T.; Repiský, M.; Noga, J.; Malkina, O. L.; Malkin, V. G. *J. Comput. Chem.* **2014**, *35*, 1725 DOI: 10.1002/jcc.23674.

(59) Dyall, K. G. *Theor. Chem. Acc.* **1998**, *99*, 366−371.

(60) Dyall, K. G. *Theor. Chem. Acc.* **2006**, *115*, 441−447.

(61) Dyall, K. G. *Theor. Chem. Acc.* **2002**, *108*, 335−340 Basis sets are available from the Dirac web site, http://dirac.chem.sdu.dk (accessed July 16, 2014).

(62) Dyall, K. G. *Theor. Chem. Acc.* **2003**, *109*, 284−284 Basis sets are available from the Dirac web site, http://dirac.chem.sdu.dk (accessed July 16, 2014).

(63) Perdew, J. P. *Phys. Rev. B* **1986**, *33*, 8822−8824.

(64) Perdew, J. P. *Phys. Rev. B* **1986**, *34*, 7406−7406.