Parallelization of a Density Functional Program for Monte-Carlo Simulation of Large Molecules

J. M. Pacheco[1] and José Luís Martins[2]

[1] Departamento de Física da Universidade, 3000 Coimbra, Portugal,
pacheco@hydra.ci.uc.pt,
WWW home page: http://aloof.fis.uc.pt/
[2] Departamento de Física, Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa, PORTUGAL
and
INESC Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
jlm@plana.inesc.pt,
WWW home page: http://bohr.inesc.pt/~ jlm

Abstract. A first-principles program designed to compute, among other quantum-mechanical observables, the total energy of a given molecule, is efficiently parallelized using MPI as the underlying communication layer. The resulting program fully distributes CPU and memory among the available processes, making it possible to perform large-scale Monte-Carlo Simulated Annealing computations of very large molecules, exceeding the limits usually attainable by similar programs.

1 Introduction

At present, an enormous effort is being dedicated to the study and fabrication of nano-structures and new materials, which calls for a framework to compute, from *first-principles*, and predict, whenever possible, properties associated with these types of systems. Among such frameworks, Density Functional Theory (DFT) constitutes one of the most promising. Indeed, the success of DFT to compute the ground-state of molecular and solid-state systems has been recognized in 1998 with the award of the Nobel Prize of Chemistry to Walter Kohn and John Pople. DFT provides a computational framework with which the properties of molecules and solids can, in certain cases, be predicted within chemical accuracy (= 1 Kcal/mol). Therefore, it is natural to try to use at profit the most recent computational paradigms in order to break new frontiers in these areas of research and development.
In this work we report the successful parallelization of an *ab-initio* DFT program, which makes use of a Gaussian basis-set. This, as will become clear in the following section, is just one of the possible ways one may write down a DFT-code. It has, however, the advantage of allowing the computation of neutral and charged molecules at an equal footing, of making it possible to write the code in a modularized fashion (leading to an almost ideal load-balance), as well as it

is taylor-made to further exploit the recent developments of the so-called order-N techniques. As a result, the program enables us to carry out the structural optimization of large molecules via a Monte-Carlo Simulated Annealing strategy.

Typically, the implementation of a molecular DFT-code using Gaussian, localized, basis-states, scales as N_{at}^3, or N_{at}^4, depending on implementation, where N_{at} is the number of atoms of the molecule. Such a scaling constitutes one of the major bottlenecks for the application of these programs to large (> 50 atoms) molecules, without resorting to dedicated supercomputers. The fact that the present implementation is written in a modular fashion makes it simple and efficient to distribute the load among the available pool of processes. **All** tasks so-distributed are performed locally in each process, and **All** data required to perform such tasks is also made available locally. Furthermore, the distribution of memory among the available processes is also done evenly, in a non-overlapping manner. In this way we optimize the performance of the code both for efficiency in CPU time as well as in **memory** requirements, which allows us to extend the range of applicability of this technique.
This paper is organized as follows: In Section II a brief summary of the underlying theoretical methods and models, as applied to molecules, is presented, in order to set the framework and illustrate the problems to overcome. In Section III the numerical implementation and strategy of parallelization is discussed, whereas in Section IV the results of applying the present program to the structural optimization of large molecules using Simulated Annealing are presented and compared to other available results. Finally, the main conclusions and future prospects are left to Section V.

2 Molecular Simulations with DFT

In the usual Born-Oppenheimer Approximation (BOA) the configuration of a molecule is defined by the positions $\boldsymbol{R}_i$ of all the N_{at} atoms of the molecule and by their respective atomic number (nuclear charge). The energy of the electronic ground state of the molecule is a function $E_{GS}(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}})$ of those nuclear positions. One of the objectives of quantum chemistry is to be able to calculate relevant parts of that function, as the determination of the full function is exceedingly difficult for all except the simplest molecules. In practice one may try to find the equilibrium configuration of the molecule, given by the minimum of E_{GS}, or one may try to do a statistical sampling of the surface at a given temperature T. That statistical sampling can be done by Molecular Dynamics (MD) or by Monte-Carlo (MC) methods. By combining the statistical sampling at a given T with a simulation process in which one begins at a high T and, after equilibrating the molecule, starts reducing the T in small steps, always equilibrating the molecule before changing T, one realizes an efficient algorithm for the global minimization of E_{GS}, the so-called Simulated Annealing Method (SAM).

The calculation of E_{GS} for a single configuration is a difficult task, as it requires the solution of an interacting many-electron quantum problem. In Kohn-

Sham DFT this is accomplished by minimizing a functional of the independent electron orbitals $\psi_i(\boldsymbol{r})$,

$$E_{GS}(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}}) = \min_{\psi_i} E_{KS}(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}}; \psi_1, \ldots, \psi_{N_{el}}) \tag{1}$$

where N_{el} is the number of electrons of the molecule, and the minimization is done under the constraint that the orbitals remain orthonormal,

$$\int \psi_i(\boldsymbol{r})\psi_j(\boldsymbol{r}) d^3r = \delta_{ij}. \tag{2}$$

The Euler-Lagrange equation associated with the minimization of the Kohn-Sham functional is similar to a one particle Schrodinger equation

$$-\frac{\hbar^2}{2m}\nabla^2\psi_i(\boldsymbol{r}) + v_{\text{eff}}(\boldsymbol{r}; \psi_1, \ldots, \psi_n)\psi_i(\boldsymbol{r}) = \epsilon_i\psi_i(\boldsymbol{r}), \tag{3}$$

except for the non-linear dependence of the effective potential v_{eff} on the orbitals. As our objective here is to discuss the numerical implementation of our algorithms, we will not discuss the explicit form of v_{eff} and the many approximations devised for its practical calculation, and just assume one can calculate v_{eff} given the electron wavefunctions $\psi_i(\boldsymbol{r})$. The reader can find the details on how to calculate v_{eff} in excellent reviews, e. g., refs.[1, 2] and references therein.

If one expands the orbitals in a finite basis-set,

$$\psi_i(\boldsymbol{r}) = \sum_j^M c_{ij}\phi_j(\boldsymbol{r}) \tag{4}$$

then our problem is reduced to the minimization of a function of the coefficients,

$$E_{GS}(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}}) \approx \min_{c_{ij}} E_{KS}(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}}; c_{11}, \ldots, c_{N_{el}M}) \tag{5}$$

and the Euler-Lagrange equation becomes a matrix equation of the form

$$\sum_j c_{ij}[H_{kj} - \epsilon_i S_{kj}] = 0 \tag{6}$$

where the eigenvalues are obtained, as usual, by solving the secular equation

$$det|H_{ij} - ES_{ij}| = 0. \tag{7}$$

The choice of the basis-set is not unique[3]. One of the most popular basis-sets uses Gaussian basis-functions

$$\phi_i(\boldsymbol{r}) = N_i \exp(-\alpha_i(\boldsymbol{r} - \boldsymbol{R}_i)^2) Z_{l(i)}^{m(i)}(\boldsymbol{r} - \boldsymbol{R}_i) \tag{8}$$

where the angular funtions Z_l^m are chosen to be real solid harmonics, and N_i are normalization factors. These functions are centered in a nucleus $\boldsymbol{R_i}$ and are an example of localized basis-sets. This is an important aspect of the method, since this implies that the matrix-elements H_{ij} result, each of them, from the contribution of a large summation of three-dimensional integrals involving basis-functions centered at different points in space. This multicenter topology involved in the computation of H_{ij} ultimately determines the scaling of the program as a function of N_{at}. Finally, one should note that, for the computation of H_{ij} one needs to know v_{eff} which in turn requires knowledge of $\psi_i(\boldsymbol{r})$. As usual the solution is obtained via a self-consistent iterative scheme, as illustrated in fig.1 .

Due to the computational costs of calculating E_{GS} from first principles, for a long time the statistical sampling of E_{GS} has been restricted to empirical or simplified representations of that function. In a seminal paper, Car and Parrinello[4] (CP) proposed a method that was so efficient that one could for the first time perform first-principles molecular dynamics simulations. Their key idea was to use molecular dynamics, not only to sample the atomic positions but also to minimize in practice the Kohn-Sham functional. Furthermore they used an efficient manipulation of the wave-functions in a plane-wave basis-set to speed up their calculations. Although nothing in the CP method is specific to a given type of basis-set, the truth is that the overwhelming number of CP simulations use a plane-wave basis-set, to the point that most people would automatically assume that a CP simulation would use a plane wave basis-set.

Although one can use plane-waves to calculate molecular properties with a super-cell method, most quantum chemists prefer the use of gaussian basis-sets. What we present here is an efficient parallel implementation of a method where the statistical sampling of the atomic positions is done with MC and the Kohn-Sham functional is directly minimized in a gaussian basis-set.

3 Numerical implementation

3.1 Construction of the matrix

Each matrix-element H_{ij} has many terms, which are usually classified by the number of different centers involved in its computation. The time and memory consuming terms are those associated with three center integrals used for the calculation of the effective potential v_{eff}. For the sake of simplicity we will assume that the effective potential is described also as a linear combination of functions $g_k(\boldsymbol{r})$,

$$v_{\text{eff}}(r, \{\psi_i\}) = \sum_{k=1}^{L} f_k(\{c_{ij}\})\, g_k(\boldsymbol{r}), \tag{9}$$

where the coeficients f_k have a dependence on the wavefunction coefficients, and g_k are atom centered gaussian functions. Actually, in the program only the exchange and correlation term of the effective potential is expanded this way, but the strategy of parallelization for all other contributions is exactly the same, and so we will not describe in detail the other terms.

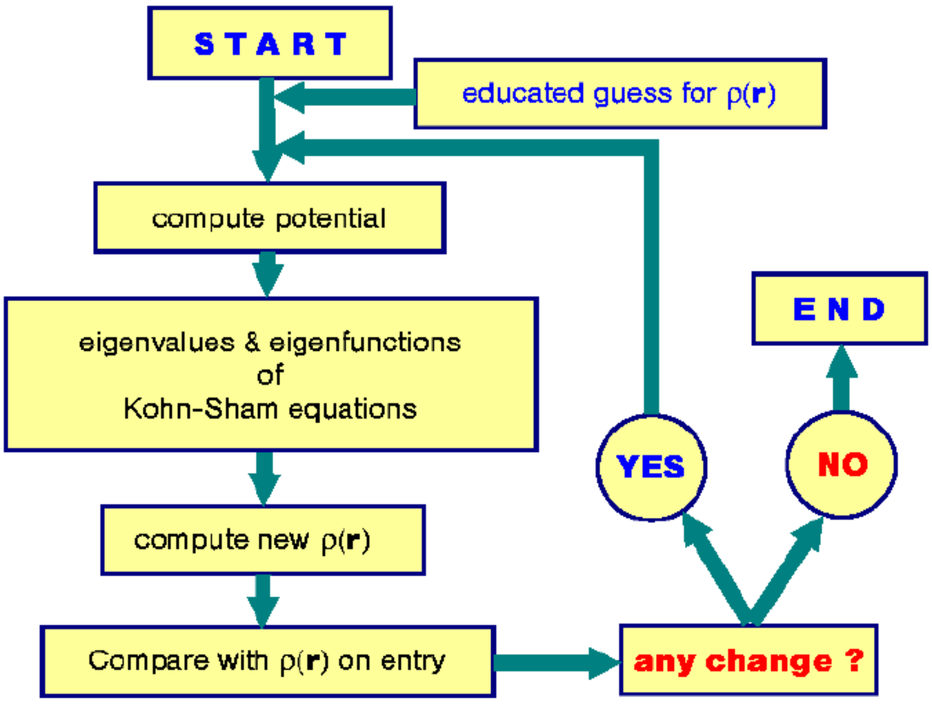

Fig. 1. self-consistent iterative scheme for solving the Kohn-Sham equations. One starts from an educated guess for the initial density which, in DFT, can be written in terms of the eigenfunctions of the Kohn-Sham equations as $\rho(\boldsymbol{r}) = \sum_i |\psi_i(\boldsymbol{r})|^2$. After several iterations one arrives at a density which does not change any more upon iteration.

The contribution of the effective potential to the hamiltonian H_{ij} is

$$\begin{aligned} V_{ij} &= \int \phi_i(\boldsymbol{r}) v_{\text{eff}}(\boldsymbol{r}, \{\psi_i\}) \phi_j(\boldsymbol{r}) d^3r = \sum_{k=1}^{L} f_k(\{c_{ij}\}) \int \phi_i(\boldsymbol{r}) g_k(\boldsymbol{r}) \phi_j(\boldsymbol{r}) d^3r \\ &= \sum_{k=1}^{L} f_k(\{c_{ij}\}) A_{ikj} \end{aligned} \tag{10}$$

where the integral $A_{ikj} = \int \phi_i(\boldsymbol{r}) g_k(\boldsymbol{r}) \phi_j(\boldsymbol{r}) d^3r$ involves three gaussian functions, and can be calculated analytically. Furthermore all dependence on wave-function coefficients is now in the coefficients f_k of the potential, and the integrals A_{ikj} are all the same in the self-consistent iterations. This means that all the iterative procedure illustrated in fig. 1 amounts now to recombine repeatedly the same integrals, but with different coefficients at different iterations throughout the self-consistent procedure.

We can now appreciate the two computational bottlenecks of a gaussian program. As the indexes i, j and k can reach to several hundred the size of the three-index array A_{ikj} requires a huge amount of memory. Although analytical, the calculation of each of the A_{ikj} is non-trivial and requires a reasonable number of floating point operations. The summation in eq. 10 has to be repeated for each of the self-consistent iterations.

So far, no parallelization has been attempted. We now use at profit the modular structure of the program in order to distribute tasks among the available processes in an even and non-overlapping way. In keeping with this discussion, we recast each matrix-element V_{ij} in the form

$$V_{ij} = \sum_{\lambda=1}^{N_{\text{proc}}} V_{ij}[\lambda] \tag{11}$$

where the indexed $V_{ij}[\lambda]$ will be evenly distributed among the N_{proc} processes executing the program, that is, it will be null except in one of the processes. Similarly, the three-index array A_{ikj} is distributed as

$$A_{ikj} = \sum_{\lambda=1}^{N_{\text{proc}}} A_{ikj}[\lambda] \tag{12}$$

in such a way that $A_{ikj}[\lambda]$ is null if $V_{ij}[\lambda]$ is null. Of course, the null elements are not stored so the large array is distributed among all the processes, which for a distributed memory machine means that A_{ikj} is distributed among all the processes. As

$$V_{ij}[\lambda] = \sum_{k=1}^{L} f_k(\{c_{ij}\}) A_{ikj}[\lambda] \tag{13}$$

there is no need to exchange the values of A_{ikj} among processes, but only those of f_k before summation, and $V_{ij}[\lambda]$ after the summation. So the calculation of

A_{ikj} is distributed among the processes, the storage is also distributed, and A_{ikj} never appears in the communications.

Finally, and due to the iterative nature of the self-consistent method, the code decides - *a priori* - which process will be responsible for the computation of a given contribution to $V_{ij}[\lambda]$. This allocation is kept unchanged throughout an entire self-consistent procedure.

3.2 Eigenvalue problem

For N_{at} atoms and, assuming that we take a basis-set of M gaussian functions per atom, our eigenvalue problem, eqs. 6 and 7, will involve a matrix of dimension $(N_{at} \times M)$. Typical numbers for an atomic cluster made out of 20 sodium atoms would be $N_{at} = 20$ and $M = 7$. This is a pretty small dimension for a matrix to be diagonalized, so the CPU effort is not associated with the eigenvalue problem but, *mostly*, with the construction of the matrix-elements H_{ij}. We have not yet parallelized this part of the code. Its paralellization, poses no conceptual difficulty, since this problem is taylor made to be dealt with by existing parallel packages, such as SCALAPACK. As this part of the code is the most CPU time consuming among the non-paralelized parts of the code, it is our next target for parallelization.

3.3 Monte-Carlo iterations

Once $E_{GS}(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}})$ is obtained for a given molecular configuration, the Monte-Carlo Simulated Annealing algorithm "decides" upon the next move. As stated before, this procedure will be repeated many thousands of times before an annealed struture is obtained, hopefully corresponding to the global minimum of E_{GS}.
When moving from one MC iteration to the next, the Simulated Annealing algorithms typically change the coordinates of one single atom $\boldsymbol{R}_\alpha \to \boldsymbol{R}_\alpha + \delta\boldsymbol{R}$. As the basis set is localized, each of the indices in A_{ijk} is associated with a given atom. If none of the indices is associated with the atom $\boldsymbol{R}_\alpha$, than A_{ijk} does not change, and therefore is not recalculated. In this way, only a fraction of the order of $1/N_{\mathrm{at}}$ of the total number of integrals A_{ijk} needs to be recalculated, leading to a substantial saving in computer time, in particular for the larger systems !
Furthermore, the "educated guess" illustrated in fig. 1, used to start the self-consistent cycle is taken, for MC iteration $n + 1$, as the self-consistent density obtained from iteration n. In this way, in *all* but the start-up MC iteration, the number of iterations required to attain self-consistency becomes small. It is this coupling between the Monte-Carlo and DFT parts of the code that allow us to have a highly efficient code which enables us to run simulations in which the self-consistent energy of a large cluster needs to be computed many thounsands of times (see below).

4 Results and discussion

The program has been written in FORTRAN 77 and we use MPI as the underlying communication layer, although a PVM translation would pose no conceptual problems. Details of the DFT part of the program in its non-parallel version have been described previously ref[6]. The MC method and the SAM algorithm are well-described in many excellent textbooks[7].

The Hardware architecture in which *all* results presented here have been obtained is assembled as a farm of 22 DEC 500/500 workstations. The nodes are connected via a fast-ethernet switch, in such a way that all nodes reside in the same virtual (and private) fast-ethernet network. In what concerns Software, the 22 workstations are running Digital Unix version **4.0-d**, the DEC Fortran compiler together with DXML-libraries, and the communication layer is provided by the free MPICH[8] distribution, version **1.1**. Nevertheless, we would like to point out that the same program has been tested successfully on a PC, a dual-Pentium II-300, running Linux-SMP, g77-Fortran and LAM-MPI[9] version **6.2b**.

We started to test the code by choosing a non-trivial molecule for which results exist, obtained with other programs and using algorithms different from the SAM. Therefore, we considered an atomic cluster made out of eight sodium atoms - Na_8. Previous DFT calculations indicate that a D_{2d} structure - left panel of fig. 3 - corresponds to the global minimum of E_{GS}[6].
Making use of our program, we have reproduced this result without difficulties. Indeed, we performed several SAM runs starting from different choices for the initial structure, and the minimum value obtained for E_{GS} corresponded, indeed, to the D_{2d} structure. One should note that one SAM run for Na_8 involves the determination of E_{GS} up to $2,2\ 10^4$ times. Typically, we have used 1000 MC-iterations at a given fixed-temperature T in a single SAM run. This number, which is reasonable for the smaller clusters, becomes too small for the larger, whenever one wants to carefully sample the phase-space associated with the $\{\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}}\}$ coordinates.

As shown in the right panel of fig. 2, Na_9^+ was our second choice. This is a nine atom sodium cluster to which one electron has been removed. As is well known[5] this cluster, together with Na_8, constitute so-called magic clusters, in the sense that they display an abnormally large stability as compared to their neighbours in size[10]. When compared with quantum-chemistry results, the DFT structures are different, both for Na_8 and Na_9^+. This is not surprising, since the underlying theoretical methods and the minimization strategies utilized are also different, at the same time that the hyper-surface corresponding to $E_{GS}(\{\boldsymbol{R}_i\})$ is very shallow in the neighbourhood of the minima, irrespective of the method. Nevertheless, recent experimental evidence seem to support the DFT results[10].

In order to test the performance of the parallelization, we chose Na_9^+ and carried out two different kinds of benchmarks. First we executed the program performing 1 iteration - the start-up iteration - for Na_9^+ and measured the CPU time T_{CPU} as a function of the number of processes N_{PROC}. For the basis-set used, the number of computed A_{ikj} elements is, in this case 328779. As can be seen from eq. 13, the ratio of computation to communications is proportional to

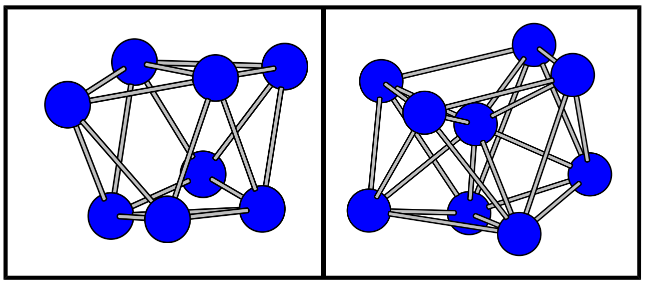

Fig. 2. global minimum of E_{GS} for the two magic sodium clusters Na_8 and Na_9^+. For the determination of such global minima a SAM algorithm has been employed, requiring many thousands of first-principles computations of E_{GS} to be carried out.

the number of fit functions L. By choosing a small molecule where L is small we are showing an unfavorable case, where the parallelization gains are small, so we can discuss the limits of our method. In fig. 3 we plot, with a solid line, the inverse of the CPU time as a function of N_{PROC}.

Our second benchmark calculation involves the computation of 100 MC-iterations. For direct comparison within the same scale, we multiplied the inverse of T_{CPU} by the number of iterations. The resulting curve is drawn with a dashed line in fig. 3.

Several features can be inferred from a direct comparison of the 2 curves. First of all, there is an ideal number N_{PROC} into which the run should be distributed. Indeed, fig. 3 shows that efficiency may actually drop as N_{PROC} is increased. For this particular system, $N_{\mathrm{PROC}} = 8$ is the ideal number. This “node-saturation” which takes place here for Na_9^+ is related to the fact that the time per iteration is small enough for one to be able to observe the overhead in communications due to the large number of nodes in which the run is distributed. When the number of atoms increases, this overhead becomes comparatively smaller and ceases to produce such a visible impact on the overall benchmarks. From fig. 3 one can also observe that, for small N_{PROC} , the largest gain of efficiency is obtained for the 1-iteration curve. This is so because that is where the parallelization plays a big role. Indeed, as stated in section 3, the number of floating point operations which are actually performed in the subsequent MC-iterations is considerably reduced, compared to those carried out during the start-up iteration. As a result, the relative gain of efficiency as N_{PROC} increases becomes smaller in this case. However, since both CPU *and* memory are distributed, it may prove convenient to distribute a given run, even if the gain is not overwhelming.

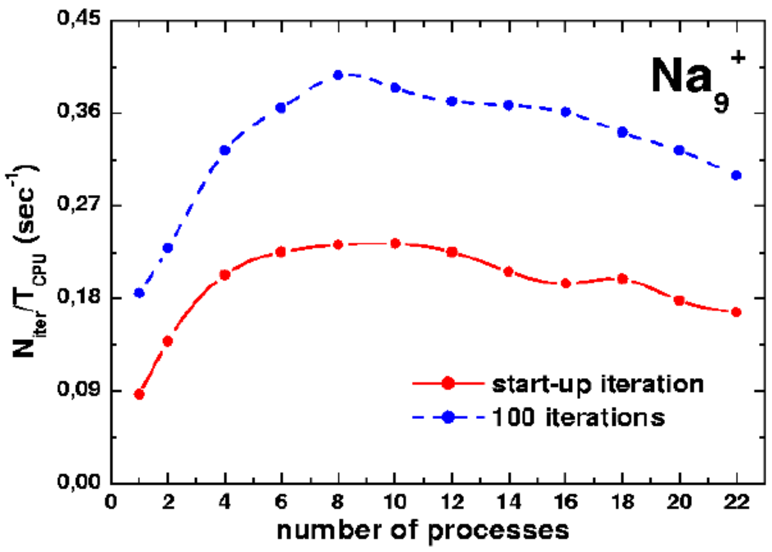

Fig. 3. Dependence of inverse CPU time (multiplied by the number of MC-iterations) as a function of the number of processes (in our case, also dedicated processors) for two benchmark calculations (see main text for details). A direct comparison of the curves illustrates what has been parallelized in the code and where the parallelization plays its major role.

The solid curve of fig. 3 is well fitted by the function $0,25 - 0,17/N_{\text{proc}}$ up to $N_{\text{proc}} = 8$ which reveals that a good level of parallelization has been obtained. This is particularly true if we consider that the sequential code has 14200 lines, and is very complex, combining many different numerical algorithms.

Finally, we would like to remark that, at present, memory requirements seem to put the strongest restrictions on the use of the code. This is so because of the peculiar behaviour of MPICH which creates, for each process, a "clone-listener" of each original process, that requires the *same* amount of memory as the original processes. This is unfortunate since it imposes, for big molecules, to set up a very large amount of swap space on the disk in order to enable MPI to operate successfully. In our opinion, this is a clear limitation. We are, at present, working on alternative ways to overcome such problems.

In fig. 4 we show our most recent results in the search for global minima of sodium clusters. The structures displayed in fig. 4 have now 21 (left panel) and 41 (right panel) sodium atoms. A total of 4147605 matrix-elements is required to compute each iteration of the self-consistent procedure for Na_{21}^+ whereas for Na_{41}^+ the corresponding number is 30787515. The structures shown in fig. 4 illustrate the possibilities of the code, which are, at present limited by swap limitations exclusively. Of course, the CPU time for these simulations is much bigger than for the smaller clusters discussed previously. In this sense, the structure shown for Na_{41}^+ cannot be considered unambiguosly converged, in the sense that more SAM runs need to be executed. On the other hand, we believe the structure depicted for Na_{21}^+ to be fully converged. Since no direct experimental data for these structures exists, only indirect evidence can support or rule out such structural optimizations. The available experimental data[10] indirectly supports this structure since, from the experimental location of the main peaks of the photo-absorption spectrum of such a cluster one may infer the principal-axes ratio of the cluster, in agreement with the prediction of fig. 4.

5 Conclusions and future applications

In summary, we have suceeded in parallelizing a DFT code which efficiently computes the total energy of a large molecule. We have managed to parallelize the most time and memory consuming parts of the program, except, as mentioned in section 3.2, the diagonalization block, which remains to be done. This is good enough for a small farm of workstations, but not for a massive parallel computer. We should point out that it is almost trivial to parallelize the Monte-Carlo algorithm. In fact as a SAM is repeated starting from different initial configurations, one just has to run several jobs simultaneously, each in its group of processors. However, this will not have the advantages of distributing the large matrix A_{ijk}. As storage is critical for larger molecules, parallelizing the DFT part of the code may be advantageous even when the gains in CPU time do not look promising.

The code is best suited for use in combination with MC-type of simulations, since we have shown that, under such circumstances, not only the results of a

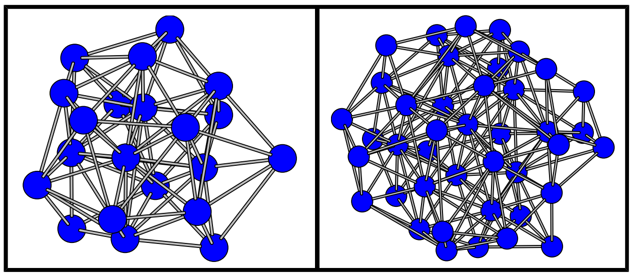

Fig. 4. Global minima for two large singly ionized sodium clusters with 21 atoms (left panel) and 41 atoms (right panel). Whereas the structure of Na_{21}^{+} can be considered as "converged", the same cannot be unambiguously stated for the structure shown for Na_{41}^{+}. For this largest cluster, the structure displayed shows our best result so-far, although further SAM runs need to be carried out.

given iteration provide an excellent starting point for the following iteration, but also the amount of computation necessary to compute the total energy at a given iteration has been worked out, to a large extent, in the previuous iteration. Preliminary results illustrate the feasibility of running first-principles, large-scale SAM simulations of big molecules, without resorting to dedicated supercomputers. Work along these lines is under way.

Acknowledgements

JMP and JLM acknowledge financial support from the Ministry of Science and Technology under contracts PRAXIS / C / FIS / 10019 / 1998 and PRAXIS / 2 / 2.1 / FIS / 26 / 94, respectively.

References

1. R.M. Dreizler, E.K.U. Gross *Density Functional Theory* (Springer-Verlag, Berlin 1990) ;
2. G. D. Mahan, K. R. Subbaswamy, *Local Density Theory of the Polarizability* (Plenum Press, 1990) ;
3. F. Alasia et al., *J. Phys.* **B27** (1994) L643 ;
4. R. Car, M. Parrinello, *Phys. Rev. Lett.* **55** (1985) 2471 ;
5. J. M. Pacheco, Walter P. Ekardt, *Annalen der Physik (Leipzig)*, **1** (1992) 254 ;
6. J. L. Martins, R. Car, J. Buttet, *J. Chem. Phys.* **78** (1983) 5646; J. L. Martins, J. Buttet, R. Car *Phys. Rev.* **B31** (1985) 1804;

7. W. M. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes* 2nd edition, (Cambridge University Press, 1990) ; H. Gould, J. Tobochnik, *An introduction to computer simulation methods*, 2nd edition, (Addison-Wesley, 1996) ;
8. W. Gropp, E. Lusk, N. Doss, A. Skjellum, *Parallel Computing*, **2** (1996) 789 ; W. Gropp, E. Lusk, *User's Guide for* `mpich`*, a Portable Implementation of MPI*, (Mathematics and Computer Science Division, Argonne National Laboratory, 1996) (http://www-unix.mcs.anl.gov/mpi/mpich/)
9. LAM Project, Laboratory for Scientific Computing, University of Notre Dame, U.S.A. (lam@mpi.nd.edu, http://www.mpi.nd.edu/lam)
10. Walter P. Ekardt (Editor), *Metal Clusters*, Wiley Series in Theoretical Chemistry, (John Wiley & Sons, 1990) ;