

# **Parallelisation of the Amsterdam Density Functional Program**

C. Fonseca Guerra, O. Visser, J. G. Snijders, G. te Velde and E.J. Baerends

Afdeling Theoretische Chemie  
Scheikundig Laboratorium der Vrije Universiteit  
De Boelelaan 1083  
1081 HV Amsterdam  
The Netherlands

1. Introduction.....	4
2. Density Functional Theory.....	7
3. Methodological Details of ADF.....	9
3.1. Solving the Kohn-Sham equation using a basis set expansion.....	9
3.1.1. Basis set.....	9
3.1.2. Numerical integration.....	10
3.1.3. Coulomb potential.....	10
3.1.4. Exchange correlation potential.....	12
3.2. Features of the ADF program.....	12
3.2.1. Frozen core .....	12
3.2.2. Construction of Symmetry orbitals .....	13
3.2.3. Fragment analyses .....	14
3.2.4. Incorporation of Relativistic Effects .....	18
3.2.5. Geometry Optimisation.....	22
3.2.6. Frequencies.....	32
4. Code structure.....	33
5. Serial performance.....	37
6. Parallelisation model.....	41
6.1. Single Program Multiple Data model.....	41
6.2. Load Balancing .....	42
6.3. Communication.....	43
7. Parallelisation of ADF.....	43
7.1. Numerical integration .....	43
7.1.1. Use of numerical integration.....	43
7.1.2. Generation of the integration points.....	45
7.1.3. Geometry optimisation .....	46
7.2. Atom pairs.....	47
7.2.1. Symmetry-unique atom pairs.....	47
7.2.2. All atom pairs.....	49
7.3. Diagonalisation.....	50
7.4. DIIS.....	51
7.5. PP Library.....	52
7.5.1. Combining incomplete matrices.....	53
7.5.2. Gathering distributed matrices.....	57

8. Parallel Performance.....	60
8.1. Timing definitions.....	60
8.2. Single point: Pt(P(Ph) <sub>3</sub> ) <sub>3</sub> CO.....	61
8.3. Gradient corrections: Cu(C <sub>7</sub> H <sub>6</sub> O <sub>2</sub> N) <sub>2</sub> .....	66
8.4. Geometry optimisation: Fe <sub>2</sub> (CO) <sub>9</sub> .....	68
9. Discussion .....	71
10. Acknowledgements.....	71
11. References.....	72

## 1. Introduction

The Amsterdam Density Functional Program (ADF) has a long history. Its development has been started at a time (the early seventies) when the full potential and implications of density functional theory, as based on the Hohenberg-Kohn theorems [1] and the Kohn-Sham molecular orbital method [2], were much less clear than today. At the time the Hartree-Fock-Slater or  $X$  method was being used extensively in solid state calculations. This is an exchange-only local-density approximation, in which the exchange energy density of a homogeneous electron gas of uniform density  $\rho_0$  is used as exchange energy density in a nonhomogeneous system (either a solid, an atom or a molecule) in an infinitesimal region  $\mathbf{r}$  surrounding a point  $\mathbf{r}$  where the density is  $\rho(\mathbf{r}) = \rho_0$ . Applications on molecular systems had been advocated by Slater [3] and had started in the late sixties with the multiple-scattering (MSX) method [4]. The MSX computational scheme uses approximations and techniques that were common in solid state physics. The muffin-tinning of the potential that was inherent to this method, precludes accurate results for molecules, notably for bond energies and molecular geometries.

The ADF program (at that time called the HFS (Hartree-Fock-Slater) program [5]) has been developed with the explicit purpose to exploit the computational advantages offered by the fact that the  $X$  model is characterised by a simple local electron potential. This property carries over to the one-electron potential of the in principle exact Kohn-Sham theory. At the same time the objective has been to construct a computational scheme that can yield accurate results, not limited by essential approximations in the computational technique. These considerations have led to the use of numerical integration, initially the approximate Diophantine scheme as adapted to molecular integrands by Ellis [6], and later more elaborate schemes, based on spatial partitioning, that can be pushed to high accuracy [7,8]. Further the traditional quantum chemical technique has been used of converting the solution of the Kohn-Sham differential equation into a matrix diagonalisation problem by the use of expansion functions. The numerical integration affords almost complete freedom in basis set choice. Slater type orbitals have therefore been chosen as basis functions. In order to avoid the classical  $n^4$  problem in the two-electron integral evaluations, an auxiliary STO basis set has been introduced to expand the electron density in atom-centred functions. Since the exchange-correlation potential can be evaluated from the electron density and its derivatives at each sample point of the numerical integration grid, the density fitting only serves to obtain the matrix elements of the Coulomb potential of the electronic charge density. Density fitting has been subsequently introduced by

Sambe and Felton [9] into DFT codes that use the traditional Gaussian basis sets of quantum chemistry and that use analytical integral evaluation instead of numerical integration to obtain the matrix elements of the effective one-electron (Kohn-Sham) hamiltonian. In this case, however, one needs an additional auxiliary basis to fit the exchange-correlation potential to make the matrix elements of this potential analytically tractable. This second fitting does require numerical integration, which therefore cannot be dispensed with. The Gaussian based techniques have been developed by Dunlap et al. [10] and have evolved into the deMon [11] and DGauss [12] codes.

Within the computational scheme sketched above, upon which we will elaborate below, one may equally well apply more advanced density functionals. This is possible due to the inherently local character of the exact exchange-correlation potential of Kohn-Sham theory and to the versatility of the numerical integration, which is suitable for the evaluation of all types of integrands that can be evaluated from functions of the local density and its derivatives. It turns out that adding correlation effects in the local-density approximation, for instance in the Vosko-Wilk-Nusair parametrisation [13] of accurate electron gas data [14], gives rather little improvement over exchange-only LDA ( $X$ ). In general at the LDA level molecular geometries and force constants are quite accurate, but bond energies are too large. However, the development of density-gradient corrections in the eighties [15,16] has led to great improvements in the accuracy of bond energies [17,18,19].

Since accurate geometries and energies are found for various types of systems, including organic molecules, coordination and organometallic compounds, heavy element compounds, crystals and surfaces, the DFT based programs offer efficient and widely applicable tools to determine the electronic structure, and the related properties, for virtually every type of system. The most important application areas will probably be those systems that due to their size are not easily amenable to highly accurate *ab initio* treatment, such as organometallic chemistry and homogeneous catalysis, coordination chemistry, surface science and heterogeneous catalysis, solid state chemistry and materials science. Application to organic molecules is of course straightforward, but in that area also highly developed *ab initio* codes are readily available.

We mention briefly some of the further developments of the ADF program system that have enhanced its usefulness and that will be discussed more extensively in the next section.

Bond energy evaluation has started with Ziegler's development of a generalised transition state method in 1977 [20]. Enhanced numerical precision in calculated bond energies has been achieved by analytical evaluation of parts of the total energy (notably electrostatic interactions between fragments [21]) and by neglecting only terms that are quadratic in the "fit error" (the

difference between true density as given by the occupied orbitals and the density described by the linear combination of fitting functions) [21,22].

The extended transition state method [20] affords an energy decomposition method [23,24]. The method is particularly interesting when the bond energy is studied between chemically realistic fragments. For this reason, and in connection with the numerical precision of the bond energy evaluation, the ADF program always builds up a molecule from fragments, the smallest possible fragment being just an atom. The fragment approach also increases the usefulness of the population analysis (see section 3.2.3).

Heavy element compounds are not only systems with many electrons (and therefore expensive), they also require proper treatment of relativistic effects. Such effects have been implemented in the ADF programs in the late seventies using a coupled-Hartree-Fock type of iterative perturbative method [25,26]. It has subsequently proven to be more accurate to apply a diagonalisation of the Pauli hamiltonian in the restricted space of nonrelativistic orbitals [27,28], which is the relativistic method that may at present be applied routinely with the ADF program. Relativistic effects are further discussed in section 3.2.4.

Automatic geometry optimisation techniques have been implemented in the ADF program system by Versluis and Ziegler [29], together with transition state searching [30,31] and frequency calculation [32], see section 3.2.5 and 3.2.6.

The evaluation of molecular properties has not yet been developed on a large scale, but will undoubtedly be an area of intense research in the near future. Infrared intensities [33], NMR [34], ESR [35] and polarisability and long-range Van der Waals interaction [36] have been the subject of recent investigations.

The ADF program package also comprises a separate code for the calculation of periodic systems, in one, two or three dimensions [37, 38,39]. The program for periodic systems will not be discussed in this paper.

In the next section some theoretical background will be given concerning the density functional theory. In section 3, the basic elements of the ADF methodology will be discussed (section 3.1) as well as a number of special features of ADF (section 3.2). In section 4 an overview of the entire code will be given and in section 5 some serial timing examples will be presented. Next we will discuss considerations concerning the parallelisation strategy (section 6) and the actual parallelisation of the code (section 7). The performance of the parallelised code is given in section 8.

The licensing information about the ADF package is obtainable by sending mail to *adf@chem.vu.nl*.

## 2. Density Functional Theory

Density functional theory (DFT) rests on the Hohenberg-Kohn theorems [1]. The first of these expresses that for systems with a nondegenerate ground state and a given electron-electron interaction there is a one-to-one mapping between the external (local) potential and the ground state wavefunction as well as the (diagonal) one-electron density. Therefore the wavefunction is uniquely determined by the one-electron density, i.e. is a functional of the density, and so are all properties, being ground state expectation values of the associated operators. In particular also the total energy and its components, such as for instance the kinetic energy, are functionals of the density. The second Hohenberg-Kohn theorem defines a density functional for the energy,  $E_v[\rho]$ , for a system with given external potential  $v$ . This energy functional will have a minimum for the exact ground state density associated with that external potential.

It has not proven possible to find sufficiently accurate approximations for  $E_v[\rho]$  so that a single Euler-Lagrange equation for the density would yield an accurate density and energy. In particular it is very difficult to find a good density functional for the kinetic energy. Therefore applications of DFT in chemistry invariably use the Kohn-Sham one-electron model [2]. Kohn and Sham introduced an auxiliary system of noninteracting electrons moving in a local potential, called  $v_s(\mathbf{r})$ . This potential has the property that the occupied Kohn-Sham orbitals yield the exact (correlated) electron density of the system. It has been proven that under certain conditions such a potential exists, and by the Hohenberg-Kohn theorem applied to noninteracting electrons, it must be unique. The important quantities in Kohn-Sham theory may now be defined as follows.

Suppose the Kohn-Sham (KS) potential  $v_s(\mathbf{r})$  were known, then the KS equations

$$h_{KS}\phi_i(\mathbf{r}) = \left[ -\frac{1}{2} \nabla^2 + v_s(\mathbf{r}) \right] \phi_i(\mathbf{r}) = \epsilon_i \phi_i(\mathbf{r}) \quad (2.1)$$

yield the exact density as

$$\rho(\mathbf{r}) = \sum_i^N |\phi_i(\mathbf{r})|^2 \quad (2.2)$$

Since the KS potential is uniquely determined by the density, so are the solutions of the KS equations. Therefore, the kinetic energy of the electrons described by the KS orbitals is a functional of the density:

$$T_s[\rho] = \sum_i^N \langle \phi_i^*[\rho](\mathbf{r}) | -\frac{1}{2} \nabla^2 | \phi_i[\rho](\mathbf{r}) \rangle \quad (2.3)$$

We may now write the exact total energy of the system as follows

$$E = T_s[\rho] + \int \rho(\mathbf{r})v(\mathbf{r})d\mathbf{r} + \frac{1}{2} \frac{\rho(\mathbf{r}_1)\rho(\mathbf{r}_2)}{r_{12}}d\mathbf{r}_1d\mathbf{r}_2 + E_{xc} \quad (2.4)$$

Here  $v$  is the external potential of the system, in our case always the nuclear potential

$$v(\mathbf{r}_1) = V_N(\mathbf{r}_1) = \sum_A \frac{-Z_A}{|\mathbf{R}_A - \mathbf{r}_1|} \quad (2.5)$$

The only unknown quantity in the equation for the total energy is the exchange-correlation energy  $E_{xc}$ , which is therefore defined by this equation. This definition of the exchange-correlation energy differs somewhat from the traditional definition as the exchange energy of the Hartree-Fock model plus the correlation energy, defined as the difference between the exact and Hartree-Fock energies. This difference is often ignored, and approximations to  $E_{xc}$  are sought that reproduce the Hartree-Fock exchange energy and the correlation energy as functionals of the density. Well-known exchange functionals are the ones introduced (as so-called generalised gradient (GGA) corrections to the LDA exchange) by Becke [16] and Perdew and Wang [40], well-known correlation functionals are the ones developed by Perdew [15], by Perdew and Wang [41] and Lee, Yang and Parr [42].

The exchange-correlation energy is clearly the crucial quantity of DFT. It is, after a Kohn-Sham calculation, the only quantity for which a reliable estimate is needed to obtain a good total energy. Moreover, it determines also the KS potential itself. It is possible to prove that the KS potential is given by

$$v_s(\mathbf{r}) = v(\mathbf{r}) + \frac{\rho(\mathbf{r})}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' + v_{xc}[\rho](\mathbf{r}) \quad (2.6)$$

where  $v_{xc}$  is the functional derivative of  $E_{xc}$

$$v_{xc}(\mathbf{r}_1) = \frac{\delta E_{xc}}{\delta \rho(\mathbf{r}_1)} \quad (2.7)$$

Unfortunately, the GGA exchange-correlation functionals, that are so successful for total energies, produce exchange-correlation potentials that have some deficiencies. Notably, the asymptotic behaviour does not have the required  $-1/r$  Coulombic form [43,44]. This results in a more or less uniform upward shift of the one-electron energies by ca. 5 eV. The density does not seem to be strongly affected, although for systems with a diffuse outer charge distribution deficiencies have been noted. In the second place the GGA potentials have a Coulombic singularity at the nucleus [44]. The sources of these deficiencies in the potentials have been identified as being caused by the so-called response part of the potential [45,46], and attempts to improve the potentials have been published [47].

We finally note that the KS scheme is an independent particle model differing from Hartree-Fock. It is connected with the *exact* solutions to the many-electron problem, in the sense that the resulting density is (in principle) the exact density and the total energy (from an accurate  $E_{XC}$ ) is in principle the exact energy. Furthermore, it has advantages for use in qualitative MO type considerations. Since the highest occupied orbital has a one-electron energy which is exactly equal to the first ionisation energy of the system, and since the virtual orbitals describe electrons moving in the field of  $N-1$  electrons and therefore lack the artificial upwards shift of the Hartree-Fock virtual orbitals, the orbital level schemes of KS calculations have properties that are usually implicitly or explicitly assumed in qualitative MO theory. This suggests that KS orbitals provide a sound basis for the application of frontier orbital considerations. In fact, the energy decomposition that has often been applied (see above) uses concepts from qualitative MO theory and gives them a quantitative basis.

### 3. Methodological Details of ADF

#### 3.1. Solving the Kohn-Sham equation using a basis set expansion.

##### 3.1.1. Basis set

The Kohn-Sham equation

$$h_{KS}\phi_i(\mathbf{r}) = \left[ -\frac{1}{2} \nabla^2 + V_N(\mathbf{r}) + V_C(\mathbf{r}) + v_{XC}(\mathbf{r}) \right] \phi_i(\mathbf{r}) = \varepsilon_i \phi_i(\mathbf{r}) \quad (3.1.1)$$

where the Coulomb potential is given by

$$V_C(\mathbf{r}_1) = \int \frac{\rho(\mathbf{r}_2)}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_2 \quad (3.1.2)$$

may be solved by expanding the solutions  $\{\phi_i(\mathbf{r}_1), i = 1, n\}$  in a set of basis functions  $\{\chi_\mu(\mathbf{r}_1), \mu = 1, m\}$ , for which in ADF Slater type orbitals are used

$$\phi_i(\mathbf{r}_1) = \sum_{\mu=1}^m C_{i\mu} \chi_\mu(\mathbf{r}_1) \quad (3.1.3)$$

With this expansion equation (3.1.1) is transformed into a secular equation from which the eigenvalues and eigenfunctions are obtained :

$$\sum_{\mu=1}^m \left[ F_{\nu\mu} - \varepsilon_i S_{\nu\mu} \right] = 0, \quad \nu = 1, m \quad (3.1.4)$$

$$\text{with } F_{\nu\mu} = \int \chi_\nu(\mathbf{r}_1) h_{KS} \chi_\mu(\mathbf{r}_1) d\mathbf{r}_1 \quad (3.1.5)$$

$$\text{and } S_{\nu\mu} = \int \chi_{\nu}(\mathbf{r}_1)\chi_{\mu}(\mathbf{r}_1)d\mathbf{r}_1 \quad (3.1.6)$$

### 3.1.2. Numerical integration

The matrix elements  $F_{\nu\mu}$  of the Fock matrix and other matrices are obtained by numerical integration[7,8]

$$F_{\nu\mu} = \sum_k w_k \chi_{\nu}(\mathbf{r}_k)h_{kS}\chi_{\mu}(\mathbf{r}_k) \quad (3.1.7)$$

where  $w_k$  is a weight factor. In order to generate a three-dimensional (3D) numerical integration grid that can integrate molecular integrands to high precision, it is necessary to take into account the cusps (due to the  $\exp(-|\mathbf{r}-\mathbf{R}|)$  behaviour of the basis functions) and singularities (due to the  $-Z/r$  nuclear potential) at the nuclear positions. This prevents one from simply generating a 3D grid over the whole molecular volume, e.g. as a product of three 1D Gauss formula's in the three Cartesian coordinates. It is necessary to partition the integration grid into subsets that use the radial variable around each nucleus in order to properly deal with the nuclear sites by the  $r^2$  factor of the Jacobian for the transformation to spherical coordinates. Complete spatial partitioning is used in ADF, either into spheres and cones [7] or into Voronoi cells [8]. It is also possible to partition the integrand into atom-centred but overlapping parts and use overlapping grids [48]. Partitioning into Voronoi cells is the standard integration scheme in ADF.

Numerical integration requires the evaluation at all grid points of the values of all basis functions and of their second derivatives to obtain  $\nabla^2\chi$ . This also allows the calculation of the density (and derivatives of the density if required) to evaluate  $v_{XC}[\rho](\mathbf{r}_k)$ . The evaluation of the nuclear Coulomb potential at each point is trivial, but this is not the case for the Coulomb potential of the electronic charge distribution, which we will discuss in the next section.

### 3.1.3. Coulomb potential

The evaluation of the Coulomb potential at grid points will be explained in more detail because of its importance for the parallelisation. Analytical evaluation of the Coulomb matrix elements would require the calculation of two-electron integrals, with the characteristic  $n^4$  problem ( $n$  is the number of basis functions). Numerical evaluation of the Coulomb matrix elements requires the evaluation of the Coulomb potential in the grid points, which leads to a large number ( $n^2p$ ,  $p$  is the number of grid points) of nuclear attraction type of integrals:

$$V_C(\mathbf{r}_k) = \frac{\rho(\mathbf{r})}{|\mathbf{r} - \mathbf{r}_k|} d\mathbf{r} = \sum_{\mu\nu} P_{\mu\nu} \frac{\chi_\mu(\mathbf{r})\chi_\nu(\mathbf{r})}{|\mathbf{r} - \mathbf{r}_k|} d\mathbf{r} \quad (3.1.8)$$

It is possible to alleviate either the  $n^4$  or the  $n^2p$  problem by approximating the exact density with an expansion in one-centre fit functions[5]

$$\rho(\mathbf{r}) = \sum_{\mu,\nu} P_{\mu\nu} \chi_\mu(\mathbf{r})\chi_\nu(\mathbf{r}) + \sum_i a_i f_i(\mathbf{r}) \quad (3.1.9)$$

The expression of the Coulomb potential in the integration point  $\mathbf{r}_k$  now becomes

$$V_C(\mathbf{r}_k) = \sum_i a_i \frac{f_i(\mathbf{r})}{|\mathbf{r} - \mathbf{r}_k|} d\mathbf{r} \quad (3.1.10)$$

To determine the fit coefficients the density is split in a sum of densities of atom pairs

$$\rho = \sum_{A,B} \rho_{AB} \quad (3.1.11)$$

$$\text{where } \rho_{AB} = \sum_{\mu \in A, \nu \in B} P_{\mu\nu} \chi_\mu^A \chi_\nu^B \quad (3.1.12)$$

and each  $\rho_{AB}$  is then approximated by an expansion of fit functions on atom A and fit functions on atom B

$$\tilde{\rho}_{AB} = \sum_i a_i f_i^A + \sum_j a_j f_j^B \quad (3.1.13)$$

Minimising the difference between the exact and the approximated density  $|\rho_{AB} - \tilde{\rho}_{AB}|^2 d\tau$  under the constraint that the number of electrons remains the same  $\rho_{AB} d\tau = \tilde{\rho}_{AB} d\tau = N_{AB}$ , leads for each atom or pair of atoms to the following set of linear equations:

$$\mathbf{S}\mathbf{a} = \mathbf{t} + \lambda \mathbf{n} \quad (3.1.14)$$

where

$$S_{ij} = \int f_i^A f_j^B d\tau \quad (3.1.15)$$

$$n_i = \int f_i^{A/B} d\tau \quad (3.1.16)$$

$$t_i = \sum_{\mu \in A, \nu \in B} P_{\mu\nu}^{AB} T_{\mu\nu i} \quad (3.1.17)$$

$$T_{\mu\nu i} = \int \chi_\mu^A \chi_\nu^B f_i^{A/B} d\tau \quad (3.1.18)$$

For the Lagrange multiplier we find from the charge preservation condition

$$\lambda = \frac{N_{AB} - \mathbf{n}^\dagger \mathbf{S}^{-1} \mathbf{t}}{\mathbf{n}^\dagger \mathbf{S}^{-1} \mathbf{n}} \quad (3.1.19)$$

As will be mentioned in the next section ADF makes use of symmetry, which for the fitting of the density implies that only the fit coefficients of the symmetry unique atoms and atom pairs are determined.

#### **3.1.4. Exchange correlation potential**

To be able to calculate the self-consistent solutions of the Kohn-Sham equation, the potential  $V_{xc}$  is derived from an approximate expression,  $\tilde{E}_{xc}$ , for the exact exchange-correlation energy,  $E_{xc}$ . Various approximations have been implemented in ADF. The exchange can be approximated by the local-density approximation (Slater's X expression[3]), with or without the non-local corrections due to Becke[16]. The correlation can be treated in the local-density approximation using the Vosko-Wilk-Nusair (VWN) parametrisation[13], also with or without non-local corrections proposed by Perdew[15]. Also the nonlocal correlation corrections of Perdew and Wang [41] have been implemented. It is possible to take the nonlocal corrections into account in a self-consistent manner, i.e. to use during the SCF cycles the  $V_{xc}$  potential that is obtained as functional derivative of the  $E_{xc}$  with nonlocal corrections. This is rather time-consuming. It is also possible to use only the LDA potential during the SCF cycles and evaluate the nonlocal corrections post-SCF, using the LDA density in the gradient corrected  $E_{xc}$ . Since the potentials corresponding to the gradient corrected  $E_{xc}$  do not show any significant improvement over the LDA potential, this seems to be a good compromise between efficiency and accuracy. During geometry optimisations with gradient corrections included, however, the full gradient-corrected potential has to be used.

### **3.2. Features of the ADF program**

Some of the characteristics of ADF have already been mentioned in the previous section; such as STO basis set, the density fit and numerical integration.

#### **3.2.1. Frozen core**

Another important aspect of ADF is the use of a frozen core. The computational time of a molecular calculation increases with the number of electrons that are treated variationally. It is therefore very important to use a frozen core especially for large molecules to minimise the valence basis set dealt with. In the frozen core approximation it is assumed that the inner-shell orbitals are unperturbed in going from the free atom to the molecular environment. Therefore the electrons populating the inner-shells of the molecule can be excluded from the variational

calculation and only the molecular orbitals of the valence electrons are used. To ensure orthogonality between the frozen core orbitals  $\{\phi_i^C\}$  and the valence basis functions  $\{\chi_v^V\}$ , the set of valence functions is made explicitly orthogonal to the frozen core. The orthogonality between the two sets is obtained by adding a STO core function  $\chi_\mu^C$  for each  $\phi_i^C$

$$\bar{\chi}_v^V = \chi_v^V + \sum_{\mu} c_{\mu v} \chi_{\mu}^C \quad (3.2.1)$$

The coefficients  $c_{\mu v}$  are obtained from the condition  $\langle \phi_i^C | \bar{\chi}_v^V \rangle = 0$ , giving

$$C = -(S^{CC})^{-1} S^{CV} \quad (3.2.2)$$

where the matrices are defined as follows:

$$C_{\mu v} = c_{\mu v} \quad (3.2.3)$$

$$S_{i\mu}^{CC} = \langle \phi_i^C | \chi_{\mu}^C \rangle \quad (3.2.4)$$

$$S_{iv}^{CV} = \langle \phi_i^C | \chi_v^V \rangle \quad (3.2.5)$$

The addition of the core functions to the basis set does not increase the size of the secular matrix, since the coefficients of the core functions are fixed by orthogonality condition. Only **F** and **S** matrices in the  $\{\bar{\chi}\}$  basis are needed. Note that the core orbitals that are implicitly used in the wavefunction, and explicitly in the calculation of the Coulomb and exchange potentials, are usually taken from very accurate atomic calculations, where  $\phi_i^C$  is a linear combination of several STO's for each core shell.

### 3.2.2. Construction of Symmetry orbitals

The ADF program makes full use of any molecular symmetry present. In particular the Fock matrix is constructed on the basis of functions that transform according to a particular irreducible representation and subspecies of the molecular point group and is consequently block diagonal and has identical blocks for the different subspecies of an irreducible representation.

The symmetry orbitals are constructed by the projection operator technique working on the primitive STO basis functions:

$$\chi_i^{\alpha} = \frac{n}{|G|} \sum_{R \in G} D_{\alpha\beta}(R^{-1}) R \chi_i \quad (3.2.6)$$

where the sum is over all symmetry operations  $R$  of the molecular point group  $G$  and  $n$  is the

dimension of the irreducible representation  $\beta$ . The irreducible representation matrices  $D_{\alpha\beta}$  are explicitly constructed in the program for all molecular point groups that have only real representations. Note that different values of  $\beta$  in the projection operator above do not necessarily lead to new linearly independent symmetry orbitals, so every newly generated combination is checked for linear independence against previously generated combinations.

Using the above procedure it is not necessary to apply the projection operator to all primitive basis functions. Rather the atoms in the molecule are divided into sets that transform into each other under the symmetry operations and from each set a representative member is chosen. Only basis functions on this representative member of each set need to be projected, since the others would not lead to new linearly independent symmetry orbitals.

The symmetry orbitals found in this way are subsequently orthogonalised against the core as described in section 3.2.1 and finally Löwdin orthogonalised among themselves. As both procedures preserve the symmetry character of the orbitals, the final set on the basis of which the Fock matrix is built forms an orthogonal set of symmetry orbitals that are also orthogonal to the core.

In a similar way the fit functions (see 3.1.3) are combined into symmetry functions, but since the total electron density has the full symmetry of the molecule, only those combinations are needed that transform according to the totally symmetric representation of the point group. Since the fitting procedure itself is done per atom pair density, which does not have the full symmetry of the molecule, the transformation to symmetry fit functions is only done at the end of the fit.

### **3.2.3. Fragment analyses**

The ADF program builds a molecule from fragments, which may either be atoms or larger moieties, e.g. ligands in a transition-metal complex, molecules in donor-acceptor molecular complexes, functional groups in organic molecules etc.. In practice this means that the results of an ADF calculation on a fragment are saved on a file for use in setting up the calculation on the overall system (the special CREATE mode is used to generate this file for the "basic" atoms). If there are symmetry-equivalent fragments (e.g. the six CO molecules in  $\text{Cr}(\text{CO})_6$ ), the program generates automatically symmetry combinations of the SCF orbitals of the fragments, using a straightforward projection technique. In each irreducible representation (irrep) of the overall symmetry group, the basis in which the KS matrix and the eigenvectors are expressed is the symmetrised set of fragment orbitals (SFO's). This offers considerable advantages when for instance orbital composition is studied, as well as charge flow in terms of

population of virtual fragment orbitals and depopulation of occupied orbitals. The SFO populations are much more meaningful numbers - in direct correspondence indeed with common chemical reasoning - than are the occupations of the primitive basis functions. It should be noted that no basis set truncation or contraction is applied, going to the SFO basis is nothing but a basis set transformation.

The use of fragments also facilitates an analysis of bond energies. It is customary to distinguish steric repulsion energy, composed of electrostatic interaction energy and Pauli repulsion, and orbital interaction (frontier orbital, HOMO-LUMO) energy. The latter may either be described as charge transfer and polarisation energies [49] or may be decomposed into contributions per irreducible representation [23,24]. A brief discussion is given here, we refer to ref. [50] for more details.

Let us consider for simplicity two interacting atoms, *A* and *B* (the results are easily generalised to molecules). The classical electrostatic energy for two nuclear charges  $Z_A$  and  $Z_B$  at positions  $\mathbf{R}_A$  and  $\mathbf{R}_B$  ( $R = |\mathbf{R}_A - \mathbf{R}_B|$ ) and electronic charge distributions  $\rho_A$  and  $\rho_B$  is

$$E_{el}(R) = \frac{Z_A Z_B e^2}{R} - \frac{Z_B e^2}{|\mathbf{r} - \mathbf{R}_B|} \rho_A(\mathbf{r}) d\mathbf{r} - \frac{Z_A e^2}{|\mathbf{r} - \mathbf{R}_A|} \rho_B(\mathbf{r}) d\mathbf{r} + \frac{e^2 \rho_A(\mathbf{r}) \rho_B(\mathbf{r})}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r} d\mathbf{r}' \quad (3.2.7)$$

We note that all four terms are of the same form (the first and fourth repulsive, the second and third attractive) and cancel at large distances (when  $\rho_A$  and  $\rho_B$  do not overlap). It is known from elementary electrostatics that the interaction energy between two interpenetrating charge clouds is smaller than for point charges, so as soon as  $\rho_A$  and  $\rho_B$  start to overlap the last term, which is repulsive, starts to decrease and the electrostatic energy is actually usually attractive for neutral systems at the distance range of interest. Only at very short distances, too short to be of chemical interest, the nuclear repulsion (first term), which becomes singular at  $R = 0$ , dominates over all other terms and causes  $E_{el}$  to become repulsive.

The repulsive character of the steric repulsion at normal bond distances clearly cannot be understood classically, and we turn to the quantummechanical description. The first approximation to the wavefunction of the interacting system is obtained by only taking into account the antisymmetry requirement (and renormalisation):

$$\Psi^0 = \frac{1}{\sqrt{2}} [\Psi_A \Psi_B - \Psi_B \Psi_A] \quad (3.2.8)$$

The total energy

$$E^0 = E^0 - E_A - E_B \quad (3.2.9)$$

is defined as the steric repulsion energy. When we split off the electrostatic part, the remainder is called the Pauli repulsion

$$E^0 = E_{\text{el}} + E^{\text{Pauli}} \quad (3.2.10)$$

Since  $E_{\text{el}}$  is usually attractive, the Pauli repulsion is responsible for the repulsive character of  $E^0$ . In order to understand the repulsive nature of  $E^{\text{Pauli}}$  it is instructive to consider the effect of forming  $\Psi^0$  on the potential and kinetic energies. Suppose, for simplicity, that A and B are one-electron systems, described by orbitals  $\varphi_A$  and  $\varphi_B$  with overlap  $S = \langle \varphi_A | \varphi_B \rangle$  (we assume  $\varphi_A$  and  $\varphi_B$  to be occupied with the same spin; an analogous result holds for doubly occupied orbitals). We obtain

$$\begin{aligned} \Psi^0(1,2) &= NA[\varphi_A(1)\varphi_B(2)] = N|\varphi_A(1)\varphi_B(2)| \\ &= \frac{1}{\sqrt{2-2S^2}} (\varphi_A(1)\varphi_B(2) - \varphi_A(2)\varphi_B(1)) \end{aligned} \quad (3.2.11)$$

(the vertical bars denote the usual "determinantal" wavefunction, i.e. they stand for antisymmetrisation and multiplication by  $1/N!$ ; the additional normalisation factor  $N$  is required since  $\varphi_A$  and  $\varphi_B$  are not orthogonal). The one-electron density corresponding to  $\Psi^0$  is

$$\rho^0(1) = 2 |\Psi^0(1,2)|^2 d\tau_2 = \frac{1}{1-S^2} \{ |\varphi_A(1)|^2 + |\varphi_B(1)|^2 - 2S\varphi_A(1)\varphi_B(2) \} \quad (3.2.12)$$

The last term shows that there is a charge depletion in the overlap region, whereas the first two terms correspond to a charge accumulation (due to the  $(1-S^2)^{-1}$  factor) in the region of the original charge distributions of A and B. We note that an alternative way of arriving at this result, which easily generalises to many-electron systems, uses the invariance of a determinantal wavefunction (up to a constant) against linear transformation of the orbitals in the determinant. If we orthogonalise  $\varphi_B$  on  $\varphi_A$ , obtaining

$$\varphi_B = \frac{1}{\sqrt{1-S^2}} (\varphi_B - S\varphi_A), \quad (3.2.13)$$

we may follow the usual rules for determinantal wavefunctions constructed from *orthogonal* orbitals, writing  $\Psi^0$  as  $|\varphi_A(1)\varphi_B(2)|$  and  $\rho^0$  as  $|\varphi_A(1)|^2 + |\varphi_B(1)|^2$ . It is easily verified that this yields the same expression for  $\rho^0$  as obtained above. For many-electron systems, with  $\Psi_A = |\varphi_1^A \varphi_2^A \dots \varphi_n^A|$  and  $\Psi_B = |\varphi_1^B \varphi_2^B \dots \varphi_m^B|$ , one obtains

$$\Psi^0 = N \left| \varphi_1^A \varphi_2^A \dots \varphi_n^A \varphi_1^B \varphi_2^B \dots \varphi_m^B \right| \quad (3.2.14)$$

where again the additional normalisation factor  $N$  is necessary because of the non-orthogonality of the  $\{\varphi_B\}$  and  $\{\varphi^B\}$  sets of orbitals. Upon carrying out an orthogonalising transformation (e.g. Gram-Schmidt)

$$\left( \varphi_1^A \varphi_2^A \dots \varphi_n^A \varphi_1^B \varphi_2^B \dots \varphi_m^B \right) = \left( \varphi_1^A \varphi_2^A \dots \varphi_n^A \varphi_1^B \varphi_2^B \dots \varphi_m^B \right) \mathbf{T} \quad (3.2.15)$$

we obtain  $\Psi^0$  in terms of a determinantal wavefunction based upon orthogonal orbitals, to which the usual Slater-Condon rules apply:

$$\left| \varphi_1^A \varphi_2^A \dots \varphi_n^A \varphi_1^B \varphi_2^B \dots \varphi_m^B \right| = \left| \varphi_1^A \varphi_2^A \dots \varphi_n^A \varphi_1^B \varphi_2^B \dots \varphi_m^B \right| \det(\mathbf{T}) = \Psi^0 \quad (3.2.16)$$

( $\det(\mathbf{T})$  is equal to  $N$ ). The density difference  $\rho^0 = \rho^0 - \rho_A - \rho_B$  corresponding to  $\Psi^0$  is characterised by a depletion area in the overlap region and accumulation of charge around the nuclei. This may be looked upon as a consequence of the Pauli principle: simply adding the probabilities  $\rho_A(\mathbf{r})$  and  $\rho_B(\mathbf{r})$  of finding an electron at a point  $\mathbf{r}$  in the bond region would fail to take into account that it is forbidden for an electron to be in the same state (at the same point, with the same spin) as any other electron.

The formation of the wavefunction  $\Psi^0$  and the concomitant charge rearrangements have the following energetic consequences. We may write the energy as the sum of the kinetic energy and the potential energy (electron-electron, electron-nucleus and nucleus-nucleus):

$$E^0 = \langle \Psi^0 | H | \Psi^0 \rangle = \langle \Psi^0 | T | \Psi^0 \rangle + \langle \Psi^0 | V | \Psi^0 \rangle = E_{\text{kin}}^0 + E_{\text{coul}}^0 \quad (3.2.17)$$

$$\text{and} \quad E^0 = E^0 - E_A - E_B = E_{\text{kin}}^0 + E_{\text{coul}}^0 \quad (3.2.18)$$

The Coulomb energy corresponding to the unmodified charge distributions  $\rho_A$  and  $\rho_B$  has been denoted as the classical electrostatic energy  $E_{\text{el}}$ . The total  $E_{\text{coul}}^0$  contains, in addition, the change in the Coulomb energy due to the density change from  $\rho_A + \rho_B$  to  $\rho^0$ , which we may denote as  $E_{\text{coul}}^{\text{Pauli}}$ :

$$E_{\text{coul}}^0 = E_{\text{el}} + E_{\text{coul}}^{\text{Pauli}} \quad (3.2.19)$$

$E_{\text{coul}}^{\text{Pauli}}$  is also stabilising, often even more so than  $E_{\text{el}}$  itself: electrons leave the internuclear region of relatively high potential and move towards regions closer to the nuclei having much lower potential. This means that the other part of  $E^{\text{Pauli}}$ ,  $E_{\text{kin}}^0$ , must be responsible for the repulsive nature of  $E^{\text{Pauli}}$ . Indeed, the kinetic energy effect of the antisymmetrisation leading to  $\Psi^0$  is strongly repulsive. One may qualitatively understand the increase in electronic kinetic

energy from the increasing gradients in  $\rho^0$  compared to  $\rho_A + \rho_B$ . These increasing gradients signify increasing kinetic energy, since the orbital gradient  $\nabla\phi$ , determining the kinetic energy

$$E_{\text{kin}} = -\frac{\hbar^2}{2m_e} \int \phi^* \nabla^2 \phi d\mathbf{r} = \frac{\hbar^2}{2m_e} \int |\nabla\phi|^2 d\mathbf{r} \quad (3.2.20)$$

is coupled to the orbital density gradient  $\nabla\rho = 2\text{Re}(\nabla\phi^* \phi)$

The ADF program prints the electrostatic energy  $E_{\text{el}}$ , calculated with the fragment SCF electron densities, and the Pauli repulsion component  $E^{\text{Pauli}}$  of  $E^0$ , as well as  $E^0$  separately:

$$\begin{aligned} E^0 &= E_{\text{el}} + E^{\text{Pauli}} \\ &= E_{\text{el}} + \left( E_{\text{kin}}^0 + E_{\text{coul}}^{\text{Pauli}} \right) \end{aligned} \quad (3.2.21)$$

The wavefunction will subsequently relax from  $\Psi^0$  to the converged ground state wavefunction  $\Psi$ . In an orbital picture this is described by mixing of virtual orbitals into the occupied (and orthogonalised) set of fragment orbitals. One may (not rigorously) distinguish mixing of virtual and occupied orbitals on one fragment (polarisation), and mixing of virtual orbitals on one fragment with occupied ones on the other (charge transfer). ADF performs a different type of analysis. The transition-state method [23,24] allows one to express the energy change in terms of the density change  $\rho - \rho^0$ , expressed in terms of the difference density matrix  $\mathbf{P} = \mathbf{P} - \mathbf{P}^0$ , and KS matrix elements in a "transition field". In its simplest form, this is just the field corresponding to the density  $(\rho + \rho^0)/2$ :

$$E_{\text{TS}} = \sum_{\mu\nu} F_{\mu\nu}^{\text{TS}} P_{\mu\nu} = \sum_{\mu\nu} F_{\mu\nu}^{\text{TS}} P_{\mu\nu} = E \quad (3.2.22)$$

The terms corresponding to the different irreps are printed by the ADF program. They are particularly useful if there different types of interactions (e.g.  $\sigma$ -donation and  $\pi$ -backdonation) occurring in different irreps.

The analysis above has been given for closed shell cases. The treatment of the formation of an electron pair bond is discussed in ref. [51].

### 3.2.4. Incorporation of Relativistic Effects

The ADF program offers the possibility of incorporating relativistic effects in two different ways, by first order perturbation theory (the so called Pauli Hamiltonian) and more recently by the so called Zeroth Order Regular Approximation (ZORA) which includes higher order effects in such a way that problems arising from the singular nature of the Coulomb potential are

circumvented, while at the same time providing a more accurate description of the relativistic effects.

**a. Pauli Hamiltonian**[25,26,28,52,53]

In this method relativistic effects are incorporated to first order in  $1/c^2$  where  $c$  is the velocity of light.

The Fock operator then has additional relativistic terms which are given by:

$$H_{Pauli} = -\frac{p^4}{8m^3c^2} + \frac{2V}{8mc^2} + \frac{1}{4mc^2} \sigma \cdot (V \times p) \quad H_{MV} + H_D + H_{SO} \quad (3.2.23)$$

Here the first term is the mass-velocity term describing the relativistic increase of the electron masses with their velocities, the second is the Darwin term, a non-classical contribution ascribed to the effect of the so called Zitterbewegung, the fast relativistic movements of the electrons around their average positions, while the final term is the spin-orbit coupling due to the interaction of the electron magnetic moments with the magnetic field generated by their own orbital motion.

The matrix elements of the first two operators, which have the same symmetry as the non-relativistic Fock operator can easily be cast into a form suitable for numerical integration:

$$\begin{aligned} \langle \chi_i | H_{MV} | \chi_j \rangle &= -\frac{\alpha^2}{8} \langle \chi_i | \chi_j \rangle \\ \langle \chi_i | H_D | \chi_j \rangle &= \frac{\alpha^2}{8} \left( \langle \chi_i | V | \chi_j \rangle + \langle \chi_i | V | \chi_j \rangle + 2 \sum_k \langle \chi_i | V | \chi_j \rangle \right) \end{aligned} \quad (3.2.24)$$

where  $\alpha = e^2/(4\pi\epsilon_0\hbar c)$  is the fine structure constant.

Computationally the extra effort involved is the calculation of the gradients of each orbital in the integration grid points, since the Laplacian was already calculated for the non-relativistic kinetic energy matrix elements.

The spin-orbit matrix elements are a little more involved since this term does not have the ordinary molecular symmetry. Instead it is invariant under the molecular point double group which affects both orbital and spin coordinates. However given orbitals that are adapted to the irreducible representations of the ordinary point group it is easy to construct spinorbitals that are adapted to the point double group by coupling space and spin part with the (double group) Clebsch-Gordan coefficients:

$$\chi_{\mu}(r,s) = \sum_{\alpha p} \chi_{\alpha}(r) \eta_p(s) \langle \alpha \ s p | \mu \rangle \quad (3.2.25)$$

where the  $\eta_p(s)$  are the primitive spinfunctions ( $\eta_1 = \alpha$   $\eta_2 = \beta$ ) and  $\Gamma_s$  is the double group representation spanned by these spinfunctions. The coupled functions are now adapted to the irreducible representation  $\Gamma$  of the double group and consequently the matrix elements of the spin-orbit operator are block diagonal in the double group irreducible representation and subspecies labels. On the basis of these functions we can write the spin-orbit matrix elements as:

$$\langle \chi_{\mu r}^1 | H_{SO} | \chi_{\mu s}^2 \rangle = \sum_{i\alpha\beta} B_i(\mu | \alpha \beta) \langle \chi_{\mu r}^1 | \chi_{\mu s}^2 \rangle \quad (3.2.26)$$

where we have separated the space and spin parts of the matrix element.

The space part is given by:

$$\langle \chi_{\mu r}^1 | \chi_{\mu s}^2 \rangle = i \frac{\alpha^2}{4} \sum_{j,k} \epsilon_{ijk} \langle \chi_{\mu r}^1 | \nabla_j | \chi_{\mu s}^2 \rangle \quad (3.2.27)$$

where  $\epsilon_{ijk}$  is the totally antisymmetric (Levi-Cevita) tensor. Clearly this matrix element again involves the gradients of the orbitals which were also needed in the mass-velocity and Darwin matrix elements and therefore involves little additional computational effort.

The spin part is finally given by:

$$B_i(\mu | \alpha \beta) = \sum_{pq} \langle \mu | \alpha \beta \rangle (\sigma_i)_{pq} \langle \mu | \alpha \beta \rangle \quad (3.2.28)$$

where the  $(\sigma_i)_{pq}$  are the usual Pauli spin matrices.

The Clebsch-Gordan coefficients  $\langle \mu | \alpha \beta \rangle$  can easily be calculated from a knowledge of the irreducible matrix representations of the double group which are explicitly generated in the program.

## b. ZORA Hamiltonian[54-60]

The use of the Pauli Hamiltonian in the above way, although quite successful for all but the heaviest elements, has some theoretical drawbacks in the presence of attractive Coulomb potentials, which can already be appreciated at the classical level. The classical relativistic energy expression for an electron in a potential  $V$  can be written as:

$$E = \sqrt{m^2 c^4 + p^2 c^2} - mc^2 + V = \frac{p^2}{2m} \left[ 1 + \frac{E - V}{2mc^2} \right]^{-1} + V \quad (3.2.29)$$

The Pauli Hamiltonian now corresponds to expansion of this expression in powers of the dimensionless quantity  $(E - V)/2mc^2$   $p^2/4m^2 c^2$  and retaining only the first order, giving the

mass-velocity correction:

$$E = V + \frac{p^2}{2m} - \frac{p^4}{8m^3 c^2} \quad (3.2.30)$$

However, close to the nuclei  $(E - V)/2mc^2$  can not be regarded as a small expansion parameter and in fact diverges in the case of an attractive Coulomb potential, which is at the root of the problems one encounters when one tries to push the Pauli expansion to higher orders and is the main reason for the relative inaccuracy of the Pauli approximation in very heavy systems. We can, however, formulate an alternative expansion that does not suffer from these problems by writing the energy expression as:

$$E = \frac{p^2}{2m} \left[ 1 + \frac{E - V}{2mc^2} \right]^{-1} + V = \frac{p^2 c^2}{2mc^2 - V} \left[ 1 + \frac{E}{2mc^2 - V} \right]^{-1} + V \quad (3.2.31)$$

When we now expand in the parameter  $E/(2mc^2 - V)$ , which remains small even close to the nuclei (in fact it goes to zero close to the nuclei) and retain only the zeroth order term we obtain

$$E = \frac{p^2 c^2}{2mc^2 - V} + V \quad (3.2.32)$$

Essentially the same idea can be applied quantummechanically on the Dirac equation, leading to the so call Zeroth Order Regular Approximation (ZORA), the Hamiltonian being given by:

$$\begin{aligned} H_{ZORA} &= \sigma \cdot p \frac{c^2}{2mc^2 - V} \sigma \cdot p + V \\ &= \sum_i p_i \frac{c^2}{2mc^2 - V} p_i + \frac{mc^2}{(2mc^2 - V)^2} \sigma \cdot (V \times p) + V \end{aligned} \quad (3.2.33)$$

Note that Hamiltonian again can be written as a sum of terms that have the usual symmetry of the molecule and a spin-orbit term that is invariant under the double group. The matrix elements of this Hamiltonian are calculated in the same way as described when discussing the Pauli Hamiltonian. In particular the spin-orbit matrix elements can again be separated into a space and a spin part, only the space part being modified by a potential dependent term. Again the only real extra effort compared to a non-relativistic calculation is the need to determine the gradients of the orbitals in the integration grid.

Experience has shown that if one is only interested in bonding energies and geometries of closed shell molecules it is usually a rather good approximation to neglect the spin-orbit terms altogether, in which case the complete calculation can be done using only the ordinary (single)

point group. Currently geometry optimisations are only implemented for these “scalar” relativistic Pauli calculations.

### 3.2.5. Geometry Optimisation

#### a. General

Geometry optimisation has been implemented in ADF by Ziegler and coworkers [61,62]. It is performed by changing the nuclear coordinates  $X$  iteratively until a minimum in the energy surface  $E(X)$  is reached. The energy can be expanded in a Taylor series around the minimum  $X_0$ :

$$E(X) = E(X_0) + g^\dagger(X_0) (X - X_0) + \frac{1}{2} (X - X_0)^\dagger H(X_0) (X - X_0) + \dots \quad (3.2.34)$$

with  $g(X)$  the vector of gradients and  $H(X)$  the Hessian

$$g_i(X) = \frac{\partial E(X)}{\partial X_i} \quad (3.2.35)$$

$$H_{ij}(X) = \frac{\partial^2 E(X)}{\partial X_i \partial X_j} \quad (3.2.36)$$

Near the minimum the higher order terms in (3.2.34) are small and therefore disregarded in the computation. In this approximation the Hessian is independent of  $X$ , so that it can be evaluated at  $X_0$ , rather than in the unknown  $X$ , and the gradients vary only linearly with  $X$

$$g(X) = g(X_0) + H (X - X_0) \quad (3.2.37)$$

Since the gradient vanishes in the minimum the step that brings us from  $X$  to  $X_0$  is

$$X - X_0 - X = -H^{-1} g(X) \quad (3.2.38)$$

If the Hessian and gradient are known in  $X$ , equation (3.2.38) yields the minimum  $X_0$ . In general this is not true exactly, due to the higher order terms in the Taylor expansion (3.2.34) that have been neglected, but as long as it is good enough to come *closer* to the minimum a repeated application of (3.2.38) will converge towards the minimum, and since the higher order terms decrease as the third power in  $(X - X_0)$ , the convergence speeds up as the minimum is approached (quadratic convergence).

#### Hessian updates

In ADF an analytical expression for the gradients is implemented, but not for the Hessian, so

that a reasonable "guess" of  $H$  must be used instead. This of course further diminishes the suitability of (3.2.38) to provide an accurate determination of  $X_0$ . To remedy this, the Hessian itself is also updated as we step along the energy surface, basically by using equation (3.2.37) and ensuring that the Hessian is modified such that it describes the changes in the gradients  $g$  in relation to the changes in  $X$ .

$$g(X_2) - g(X_1) = H (X_2 - X_1) \quad (3.2.39)$$

In a one-dimensional space equation (3.2.39) would uniquely define  $H$  but this is not the case in more dimensions. Well-established methods are known to deal with the degrees of freedom that remain and several of them are available in ADF.

### DIIS

While we step along in search for the minimum, the updates of the Hessian gradually build into it information about the (local) curvature of the energy surface, hopefully to improve the steps that we are going to take. However, although the update scheme ensures that equation (3.2.39) holds for the step just taken and the observed change in the gradient, it does not, in general, preserve this condition in the update that will be carried out after the next step. Some of the old information may get lost in this way and we do not exploit fully all the information that is available.

A powerful method that takes the history better into account, known by the acronym DIIS (Direct Inversion in the Iterative Subspace) [63] can be applied to geometry optimisations [64] and has been built into ADF. Write the coordinate vector  $X$  as  $X_0$  plus a deviation  $\delta X$  and consider a linear combination of (previous) coordinate vectors with the sum of combination coefficients equal to one

$$X^* = \sum_i c_i X_i = X_0 + \sum_i c_i \delta X_i \quad (3.2.40)$$

The DIIS method consists of choosing the coefficients  $c_i$  such that the deviation  $\delta X^*$  is minimised. In combination with the constraint (sum of coefficients must equal one), this gives the system of linear equations

$$c_i = 1 \quad (3.2.41a)$$

$$\text{and } \frac{\partial}{\partial c_j} \left\{ \left( \sum_i c_i \delta X_i \right) \left( \sum_i c_i \delta X_i \right) + 2\lambda \sum_i c_i \right\} = 0, \quad j = 1, 2, \dots \quad (3.2.41b)$$

$$\text{hence } \delta X_j \left( c_j \delta X_j \right) + \lambda = 0, \quad j = 1, 2, \dots \quad (3.2.41c)$$

The true deviations  $\delta X_i$  are of course unknown. Our best estimate for them is provided by equation (3.2.37) and is used therefore. So in (3.2.41) we set

$$\delta X_i = H^{-1} g_i \quad (3.2.42)$$

Using our current best estimate for the Hessian  $H$  and a sequence of previous coordinate vectors  $X_i$  with corresponding gradients  $g_i$ , we solve (3.2.41) and obtain the coefficients  $c_i$  and hence  $X^*$ , the combination of coordinate vectors with the smallest expected deviation from  $X_0$ .

Our best estimate for the gradient  $g^*$  in  $X^*$  is not zero but, as can be derived from (3.2.37) or (3.2.42)

$$g^* = H \delta X^* = c_i g_i \quad (3.2.43)$$

We *assume* now that the gradient in  $X^*$  is actually given by (3.2.43) and apply the normal procedure to take a step towards the minimum from a given  $X$  with corresponding gradient, in this case:

$$X^* = -H^{-1} g^* \quad (3.2.44)$$

In this set-up we obtain therefore two steps, the first is the DIIS step,  $X^* - X$ , the second is  $X^*$  (3.2.44) and the total step we actually take from the most recent point  $X$  is the sum of both.

### Summary of Optimisation Scheme

From the foregoing we obtain the following optimisation scheme:

1. For a given nuclear frame  $X$ , solve the electronic SCF problem and compute from its solution the energy gradients  $E / X$
2. By comparison with the gradients obtained for the previous  $X$ , improve the approximation of the Hessian using an update scheme based on (3.2.39)
3. With the improved Hessian, estimate for a sequence of previous coordinate vectors  $X_i$  the associated deviations  $\delta X_i$  (3.2.42)
4. Solve the DIIS equations to get  $X^*$  and the corresponding gradient  $g^*$  (3.2.41, 3.2.43).
3. apply (3.2.44) to compute a step  $X^*$  and take  $X^* + X^*$  as the next approximation for  $X_0$
4. if not converged return to step 1.

Convergence is defined by three quantities that must fall below user-specified thresholds: the

variation in energy near  $X$ , the gradient at  $X$  and the program-estimated "error" at  $X$ : the step computed at  $X$  to obtain the next guess for  $X_0$ .

Optimisations can be done either in cartesian or in internal (Z-matrix) coordinates. Restricted optimisations are carried out by "freezing" one or more nuclear coordinates: in the scheme above, the steps for the frozen coordinates are simply set to zero and in the convergence tests the gradient components that correspond to the frozen coordinates are left out.

## b. Hessian

### Initial Guess

To start the geometry optimisation scheme an initial guess for the Hessian is required. In ADF this initial Hessian is derived from parametrised empirical force constants for all stretches, bends, torsions and out-of-plane coordinates formed by atoms that are close to each other, taken from Fisher and Almlöf [64]. In general there are more such coordinates than the  $3N-6$  molecular degrees of freedom, which is why they are called *redundant* internal coordinates. The initial Hessian in the representation of these coordinates is diagonal, with the diagonal elements given by the force constants. The Hessian in the *cartesian* nuclear coordinates is then

$$H^{cart} = B^{red\dagger} H^{red} B^{red} \quad (3.2.45)$$

where  $B$  contains the transformations between internal and cartesian coordinates.

$H^{cart}$  is a  $3N \times 3N$  matrix with 6 zero eigenvalues (corresponding to the 3 rotations and 3 translations, which leave the energy unchanged).

If the geometry optimisation is to be carried out in user-defined internal coordinates (Z-matrix) rather than in cartesians, the corresponding Hessian is obtained as

$$H^{internal} = B^{-1\dagger} H^{cart} B^{-1} \quad (3.2.46)$$

### Update Schemes

To improve the Hessian given an observed change  $g$  in the gradient resulting from an applied change  $X$  in the nuclear coordinates, a correction term is constructed. In view of the repeatedly needed application of (3.2.38), the program processes the *inverse* of the Hessian, rather than the Hessian itself. Let this inverse be denoted  $S$ . The update scheme is then

$$S_k = S_{k-1} + S^{corr} \quad (3.2.47)$$

$S^{corr}$  is chosen such that

$$X_k = S_k \quad g_k \tag{3.2.48}$$

Since the Hessian and hence  $S$  and  $S^{corr}$  are symmetric we can generally express

$$S^{corr} = c_1 a_1 \times a_1^\dagger + c_2 a_2 \times a_2^\dagger + \dots \tag{3.2.49}$$

where  $a_1, a_2, \dots$  are vectors and  $c_1, c_2, \dots$  constants. A formula that uses only the first term in (3.2.49) is called a "rank one" update formula. Including also the second term yields a rank two formula (etc.).

A rank one formula is completely defined by (3.2.48). This is known as the Murtagh-Sargent formula [65]:

$$\begin{aligned} a &= X - S_{k-1} \quad g \\ c &= \frac{1}{a \quad g} \end{aligned} \tag{3.2.50}$$

It has the intuitively attractive property that it is strictly determined by the imposed conditions: in a sense it is the smallest adaptation of the Hessian to make it satisfy equation (3.2.48). However, although the Murtagh-Sargent method has been widely used for minimisations, it does not always preserve the positive-definiteness of the Hessian, which is highly desirable, and in addition the constant  $c$  (3.2.50) may blow up, resulting in numerical problems.

More robust and stable update schemes are provided by rank two formulas. Now, however,  $S_k$  and hence  $S^{corr}$  are not defined uniquely anymore. In fact there is a large number of alternatives, but the number of possibilities reduces considerably if the restriction is imposed that  $S^{corr}$  be defined in terms of  $S_{k-1}$ ,  $g$  and  $X$  only, which seems fairly natural. Most update schemes are conveniently written as

$$\begin{aligned} S^{corr} &= d_1 X \times X^\dagger + d_2 (S \quad g) \times (S \quad g)^\dagger \\ &\quad + d_3 \{ X \times (S \quad g)^\dagger + (S \quad g) \times X^\dagger \} \end{aligned} \tag{3.2.51}$$

The precise choice is to some extent a matter of taste. Particular choices may work better in some cases than in others.

One of the most popular update schemes is provided by the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula [66-69], claimed to have high quality as regards stability and convergence to the true Hessian. One of its attractive properties is that it has a bias towards positive eigenvalues of the Hessian: with some marginal precautions positive definiteness is retained through the updates [70]. This is rather relevant as it can be seen from (3.2.38) that the

step would steer us uphill along eigenvectors of the Hessian that have negative eigenvalues.

$$\begin{aligned}
 d_1 &= \frac{X \quad g + \quad g \quad S \quad g}{(X \quad g)^2} \\
 d_2 &= 0 \\
 d_3 &= \frac{-1}{X \quad g}
 \end{aligned}
 \tag{3.2.52}$$

Another well-know variety is the Davidson-Fletcher-Powell (DFP) formula (for a derivation see [71]):

$$\begin{aligned}
 d_1 &= \frac{1}{X \quad g} \\
 d_2 &= \frac{-1}{g \quad S \quad g} \\
 d_3 &= 0
 \end{aligned}
 \tag{3.2.53}$$

By default the BFGS formula is used in ADF. The user can override this and choose another update scheme. The available alternatives are: Murtagh-Sargent (equation 3.2.50), Davidson-Fletcher-Powell (3.2.53), and the Fletcher-switch and Hoshino formulas. The latter two are expressed in constants  $k_i$

$$\begin{aligned}
 k_1 &= \frac{1}{\sqrt{g \quad g}} \\
 k_2 &= \frac{1}{g \quad S \quad g} \\
 k_3 &= k_1 \quad 1 + \frac{k_1}{k_2}
 \end{aligned}
 \tag{3.2.54}$$

The Fletcher-switch method gives for the update formula (see equation 3.2.51):

$$d_1 = k_3, \quad d_2 = 0, \quad d_3 = k_1
 \tag{3.2.55a}$$

if  $k_2$  is greater than  $k_1$  and otherwise

$$d_1 = k_1, \quad d_2 = k_2, \quad d_3 = 0
 \tag{3.2.55b}$$

The Hoshino formula is

$$\begin{aligned}
 d_1 &= k_1(1 - \phi) + k_3\phi \\
 d_2 &= -k_2(1 - \phi) \\
 d_3 &= -k_1\phi
 \end{aligned}
 \tag{3.2.56a}$$

$$\text{with } \phi = \frac{k_2}{k_2 + k_1}, \text{ respectively } \frac{k_2}{k_2 - k_1} \quad (3.2.56b)$$

if only one of these values is in the range (0,1), the maximum of the two if both are in that range, and  $\phi = 1$  otherwise.

Yet another update scheme for the Hessian is implemented in ADF: the Powell formula. Contrary to for instance the BFGS scheme it does not have the property that it preserves the positive definiteness of the Hessian. This property is highly desirable in minimisations but certainly not in Transition State searches (see below) where at the transition state itself one of the Hessian eigenvalues is negative.

In ADF the Powell scheme can therefore not be chosen in a minimisation, but it is very suitable for a Transition State search and is indeed the default method in that application. In fact, in a TS search the only alternative that the user can opt for in the current version of ADF is the BFGS scheme (but that is not recommended in view of its bias towards positive Hessian eigenvalues). The Powell update formula is (expressed now in the Hessian, rather than its inverse)

$$H_k = H_{k-1} + H^{corr} \quad (3.2.57a)$$

$$H^{corr} = \frac{g \times X^\dagger + X \times g^\dagger - \frac{g}{X} \frac{X}{X} X \times X^\dagger}{X \quad X} \quad (3.2.57b)$$

### c. Gradients

The energy gradients are computed from the self-consistent molecular orbitals, the solutions of the electronic SCF equations for the given nuclear frame X [61,62]. A straightforward differentiation of the energy expression with respect to nuclear displacements yields a (relatively simple) expression which contains large terms that analytically cancel but that may not cancel exactly in the numerical integrations that are used to evaluate them. To avoid any numerical problems in this respect the energy derivative is rewritten to explicitly eliminate such terms. As a consequence the final expression which is actually used in the program looks a bit more complicated and takes also somewhat more computing time to evaluate.

The implemented formula for the evaluation of the energy derivative with respect to a nuclear displacement reads

$$\frac{E}{X_A} = \int \frac{\partial}{\partial X_A} \{T - \epsilon_i\} dr$$

$$\begin{aligned}
& - \int \frac{1}{X_B} \{T - \epsilon_i\} \psi_i^A dr \\
& + 2 \int \frac{1}{X_A} \left\{ (V_N^B + V_C^B) + V_C \right\} \psi_i dr \\
& - 2 \int \frac{1}{X_B} \left\{ (V_N^A + V_C^A) \right\} \psi_i dr \\
& - \sum_k n_k \frac{\omega_k^A}{X_A} \left\{ (V_N^B + V_C^B) + V_C \right\} \omega_k^A dr \\
& + 2 \sum_k n_k \frac{\omega_k^B}{X_B} \left\{ V_N^A + V_C^A \right\} \omega_k^B dr \\
& + 2 \sum_{\gamma} \frac{1}{X_A} \left\{ V_{XC}^\gamma \psi_i - V_{XC}^{\gamma(A)} \psi_i^A \right\} dr \\
& + \sum_{\gamma} \frac{\rho_c^A}{X_A} \left\{ V_{XC}^\gamma - V_{XC}^{\gamma(A)} \right\} dr \\
& + \sum_{B < A} \frac{V_{AB}}{X_A} \\
& - 2 \sum_{\gamma} \sum_{\mu} \sum_{\tau} c'_{i\mu} F_{\tau i}^\gamma S_{\zeta\tau}^{-1} \left\{ \frac{\chi_\mu}{X_A} \omega_\zeta - \frac{\chi_\mu}{X_B} \omega_\zeta^A \right\} dr \\
& + 2 \sum_{i} \sum_{\mu} \sum_{\tau} c'_{i\mu} \epsilon_i S_{\tau i} S_{\zeta\tau}^{-1} \left\{ \frac{\chi_\mu}{X_A} \omega_\zeta - \frac{\chi_\mu}{X_B} \omega_\zeta^A \right\} dr
\end{aligned} \tag{3.2.58}$$

$X_A$  is a coordinate of nucleus A;

$\psi_i$  is an occupied molecular valence orbital, expanded in atom-centred Slater-type basis functions;

$\psi_i^A$  is the part of  $\psi_i$  that consists of the Slater-type basis functions on atom A

$\epsilon_i$  if the eigenvalue of  $\psi_i$ ;

$V_N^A$  is the nuclear potential of atom A;

$V_C^A$  is the electronic Coulomb potential due to the (spherically symmetric) charge density of atom A;

$V_C$  is the electronic Coulomb potential due to the "deviation" density: the SCF density minus

the sum of spherical atom densities;

$V_{AB}$  is the electrostatic energy of the atom pair AB:  $(Z^A/r + V^A(r))(Z^B\delta(r - R_B) + \rho^B(r))dr$

$\omega_k^A$  is a frozen core orbital of atom A with occupation number  $n_k$ ;

$\gamma$  runs over the two spins

$V_{XC}^\gamma$  is the exchange-correlation (density functional) potential for spin  $\gamma$  electrons due to the molecular SCF charge density;

$V_{XC}^{\gamma(A)}$  is the exchange-correlation potential for spin  $\gamma$  due to the charge density of atom A only.

$\rho_c^A$  is the frozen core density of atom A;

$\chi_\mu$  is a core *function* (included in the valence basis set for orthogonalisation of the true valence basis functions on the frozen core *orbitals*, see section 3.2.1)

$S$  is the (square) overlap matrix of core *functions* with core *orbitals*.

$S_{\tau i}^n$  is the overlap of core function  $\tau$  with  $i$ ;

$F_{\tau i}^n$  is the Fock matrix element between core function  $\tau$  with  $i$ ;

$c_{i\mu}$  is the coefficient in  $i$  of the  $\mu$ -the valence basis function orthogonalised on all frozen core orbitals.

All integrals in (3.2.58) are computed by numerical integration.

#### d. Transition States

Transition States are of high chemical interest, but computationally often hard to pin-point. In addition the quality of results may be very critical when for instance the energy barrier between reactants and product is required at kcal precision. For applications of density-functional theory, notably with ADF, to Transition State searches see for instance [72].

From a simple mathematical point of view, the determination of a Transition State (TS) is very similar to a minimisation. A stationary point is searched where the gradient vector vanishes. An initial Hessian is updated in the course of the optimisation iterations and the DIIS technique is used to help convergence.

The difference between a minimum and a TS is that at the TS the Hessian has one negative eigenvalue (for higher order Transition States there are more negative eigenvalues). When we are close enough to the TS, which means basically that the Hessian (and our estimate for it) has the correct number of negative eigenvalues and that the anharmonic terms are small, exactly the same methods can be used as for a minimisation. The problem is, however, that the energy surface around Transition States very often displays a large anharmonicity. An approach based on neglecting the third- and higher order terms therefore runs the risk to fail unless we start

very close to the TS: if we take steps that are too big we easily find ourselves in a region that has only positive eigenvalues – we jumped over the hill too far – so that simply continuing our usual method will bring us to the nearest local minimum, rather than (back) to the Transition State.

The aspects that discriminate TS searches from minimisations are largely related to these problems. A different Hessian update scheme is applied (which will not attempt to make or keep all eigenvalues positive), control by the program of the maximum step lengths to take is stricter, and checks are carried out that the Hessian has the right structure and, if it is found to be wrong, the eigenvalues are shifted (temporarily) to enforce the right direction of the step to take. The procedure used in ADF [72] is based on the Rational Function approach [73].

### **e. Linear Transit**

One of the options in ADF is to trace a path on the energy surface. The path is defined by a systematic variation of one or more nuclear coordinates between user-specified initial and final values in a user-specified number of steps. The coordinates that are so varied are denoted the Linear Transit (LT) parameters. If there is more than one LT parameter, all such parameters are modified simultaneously. It is not possible in ADF to scan a two-dimensional (or even higher dimensional) surface by *independently* varying such parameters. A Linear Transit is therefore often called a *Synchronous* Linear Transit.

For each LT "point", defined by a particular value of the path parameter(s), the other nuclear coordinates that are not LT parameters can be optimised (or a subset of them). From the program's point of view a LT run is simply a sequence of (constraint) optimisations.

One of the most useful applications of a LT is to obtain a good first guess of a transition state. A classical example is the HCN to CNH transit, in which the Hydrogen is moved from one end of CN to the other. The LT parameter would here be the angle  $\angle(\text{H-C-N})$ , changing from 180 degrees down to zero in 10 steps say. For each of the  $\angle$ -values the other geometric variables (the HC and CN distances) are optimised. By gathering the energy values in the LT points, and interpolating the results in a curve, a fair estimate can be obtained of the transition state between the two energy minima in the endpoints of the path.

Of course, when we know that the transition state is probably not very far from  $\angle = 90$ , it is computationally advantageous to use a smaller interval for the LT path, for instance from 50 to 130 degrees, and reduce the number of LT points accordingly.

### 3.2.6. Frequencies

Since ADF has not (yet) an implementation of analytical second derivatives, the computation of frequencies is carried out by slightly displacing nuclear coordinates from their equilibrium value and computing the corresponding force constants by numerical differentiation of the gradients. The considerable computational effort that is implicated by this approach – a large number of SCF calculations – is reduced for molecules with symmetry by making use of symmetry relations between displacements [33].

The matrix of force constants is set up in the representation of mass-weighted cartesian coordinates:

$$F_{ij} = \frac{\partial^2 E}{\partial q_i \partial q_j} \quad (3.2.59)$$

$$\begin{aligned} q_1 &= X^{(1)} \sqrt{m^{(1)}} \\ q_2 &= Y^{(1)} \sqrt{m^{(1)}} \\ \text{with } q_3 &= Z^{(1)} \sqrt{m^{(1)}} \\ q_4 &= X^{(2)} \sqrt{m^{(2)}} \\ &\text{etc..} \end{aligned} \quad (3.2.60)$$

Diagonalisation of  $F$  yields the normal modes as eigenvectors. The eigenvalues are directly related to the frequencies:

$$\nu_k = \frac{\sqrt{\lambda_k}}{2} \quad (3.2.61)$$

When building the force constants matrix one can use cartesian or internal (Z-matrix) displacements. In both cases the final matrix is transformed to mass-weighted cartesians and then diagonalised to yield the frequencies and normal modes. The latter can then be transformed back to a representation in internal coordinates if required.

With ADF it is possible to run a frequencies calculation where only a subset of the nuclear coordinates is displaced, thereby neglecting any coupling with the other coordinates. This is a fast way to study some well-selected normal modes if one knows that the omitted displacement will hardly contribute to the mode(s) one is interested in.

### IR intensities

A by-product of the SCF calculations in the displaced geometries are the dipole moments, so that the dipole derivatives can be computed by numerical differentiation. From these the

infrared intensities are obtained:

$$A_i = 974.86 g_i \frac{\partial \mu}{\partial Q_i}^2 \quad (3.2.62)$$

$A_i$  is the absorption intensity in [km/mol] of normal mode  $Q_i$ ;  $g_i$  is the degeneracy of the mode;  $\mu$  is the dipole moment in atomic units.  $Q_i$  is the normal mode (mass weighted cartesian in atomic units; masses in a.m.u.).

In the evaluation and processing of gradients care is taken to remove incorrect contributions from rotations and translations which may spuriously show up due to the finite precision of the numerical integrations and other inaccuracies. The energy must be invariant under such rigid movements.

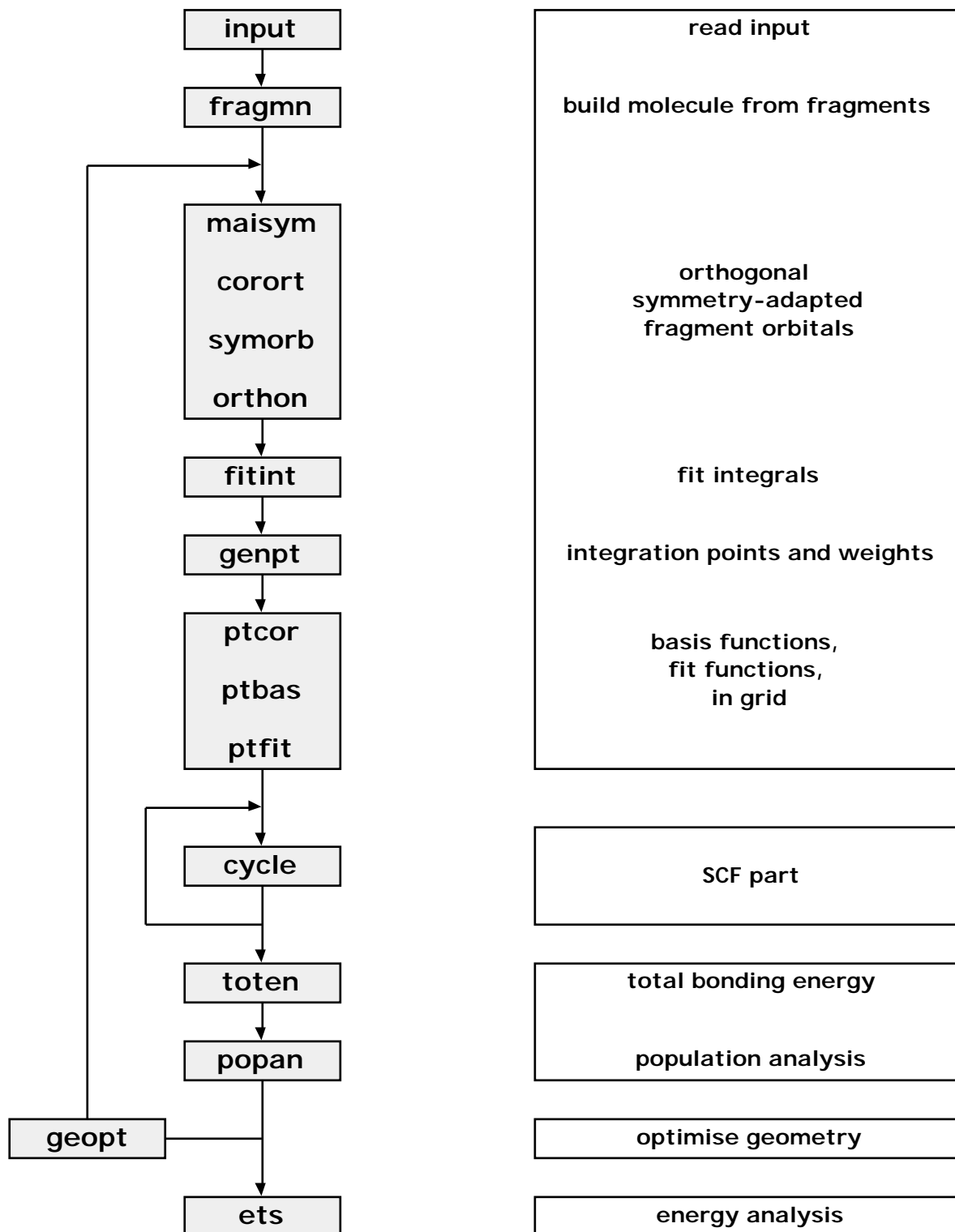
This does not hold for the dipole derivatives: they are invariant under translations but rotations may contribute. The effects of the three rotations on the equilibrium dipole moment are determined first. Then for each cartesian displacement  $\delta q$ , its three rotational components are determined and hence the rotational components in  $\mu / q$ .

#### 4. Code structure

For a single point calculation (calculation at a fixed geometry) the ADF program consists of four parts as we can see from figure 1 where the flow of the program is represented. In the first part the input is read and then the numerical data to be used in the SCF part is prepared, the next part contains the SCF and in the last part the population and energy analysis is done.

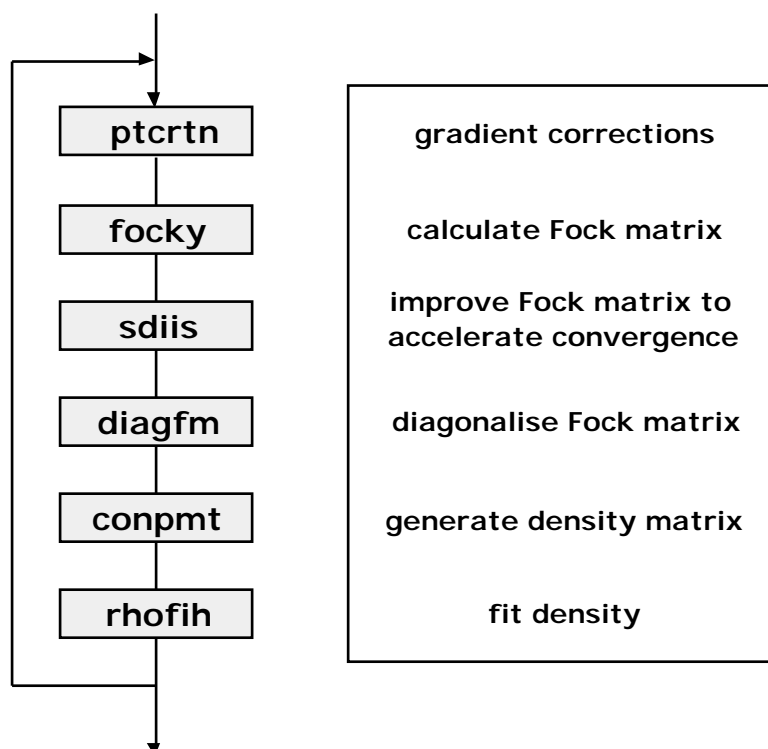
If we look into more detail we see that several set-ups are done in the second part before the SCF. After the input has been read and the molecule has been built from the (atomic or molecular) fragments, the fragment orbitals are orthogonalised to the core (CORORT), symmetrised (SYMORB) with symmetry information from MAISYM and then mutually orthogonalised (ORTHON) to construct the orthogonal symmetry-adapted fragment orbitals.

For the density fit the fit integrals (3.1.15), (3.1.16) and (3.1.18) are precomputed by the subroutine FITINT and written to file. Then the program generates the integration points and the weights of the integration points. The last three routines of this part calculate the value of the core density, the core Coulomb potential, the basis functions, fit functions and the fit Coulomb potential in the integration points and write results to file.



*Figure 1 Flow of ADF program.*

To be able to do calculations on large molecules a direct SCF option has been included in ADF. Instead of reading the values of the functions in the integration points from file, they are recalculated each iteration during the SCF. This saves a lot of disk space, but it also costs much more CPU time. All the calculations described in this paper use the direct SCF only for the fit functions. The values of the basis functions in the integration points are calculated once and written to file.



*Figure 2 The SCF part of ADF*

The SCF part is handled by CYCLE. It consists of six major subroutines (see figure 2). In FOCKY the Fock matrix is calculated on the orthogonal symmetry-adapted basis set.

To accelerate the SCF convergence the ADF program uses DIIS (Direct Inversion of Iterative Subspace)[63] The essence of the DIIS algorithm is that the results of the previous cycles are used to make a better guess for the new Fock matrix. This DIIS algorithm calculates in each cycle the error vector  $\mathbf{e}_i$ , which is the commutator of the current Fock matrix and the current density matrix on the orthogonal symmetry adapted and core-orthogonalised basis set. Then the residuum vector, which is a linear combination of the previous and current error vectors has to approximate the zero vector in the mean-square sense under the condition that the expansion coefficients sum up to one. This leads to a system of linear equations

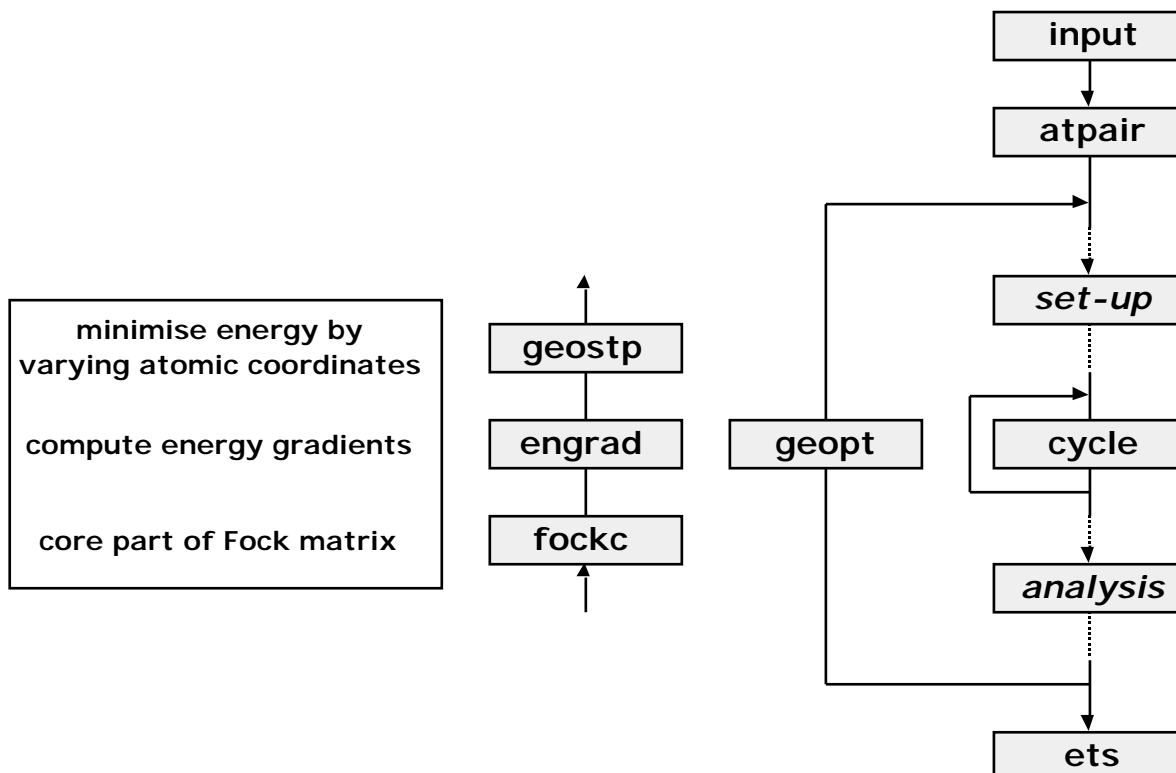
$$\begin{array}{ccccccc}
B_{11} & B_{12} & \dots & B_{1m} & -1 & c_1 & 0 \\
B_{21} & B_{22} & \dots & B_{2m} & -1 & c_2 & 0 \\
\dots & \dots & \dots & \dots & \dots & \dots & = 0 \\
B_{m1} & B_{m2} & \dots & B_{mm} & -1 & c_m & 0 \\
-1 & -1 & -1 & -1 & 0 & & -1
\end{array} \tag{4.1}$$

which are solved by Cholesky decomposition. This matrix  $B_{ij} = \langle \mathbf{e}_i | \mathbf{e}_j \rangle$  is called the overlap matrix of the error vectors. Then the "extrapolated" Fock matrix can be calculated as a linear combination of the Fock matrices of previous cycles. The routine DIAGFM diagonalises this Fock matrix.

Finally the subroutine CONPMT generates the density matrix on the primitive basis set from the calculated eigenvectors to enable RHOFIH to determine the fit coefficients for the new density. RHOFIH reads for each atom pair from file the integrals (3.1.18) and calculates from equation (3.1.14) the coefficients for that atom pair. This cycle from FOCKY to RHOFIH is repeated until convergence has been reached.

As mentioned in the previous section the quality of the ADF calculations can be improved by adding GGA (Generalised Gradient Approximation) to the exchange-correlation potential during the SCF. It is also possible to use the gradient corrections post-SCF. The total energy with the gradient corrections is calculated from the LDA density. These density gradient corrections depend at a certain point on the first and second derivative of the density at that same point. So to evaluate the Fock matrix the value of the GGA potential in the integration points is needed. Therefore the derivatives of the fit functions in the integration points are calculated and knowledge of the fit coefficients makes it then possible to calculate the numerical value of these corrections in all integration points. The correction to the XC-potential is calculated by the routine PTCRTN, which is called before FOCKY in CYCLE. The value of the derivatives of the fit functions in the integration points is recalculated each iteration and the corrections in each integration point are written to file.

After convergence is reached the population analysis and decomposition of the energy is done. TOTEN computes by numerical integration energy terms related to the following densities: the sum of fragment densities; the density  $\rho^0$  of orthogonalised fragments (see section 3.2.3); and the SCF density. Then POPAN performs a Mulliken population analysis on the basis of the primitive STO's (if required for individual orbitals) and calculates the atom-atom population matrix and the charge of the atoms. The last routine ETS calculates interaction energy and performs the population analysis in terms of symmetrised fragment orbitals.



*Figure 3 Flow of geometry optimisation in ADF program.*

In figure 3 (see also figure 1) a flow of the program for a geometry optimisation is shown. After reading the input the routine ATPAIR is called once to calculate the atom-pair electrostatic interactions for geometry updates. Then the geometry cycles are started.

The routine that handles the geometry optimisation is GEOPT. After the population analysis has been done ADF calls GEOPT to generate the new atomic co-ordinates. Then ADF performs the whole set-up from MAISYM to PTFIT again, it calculates the new converged energy and performs the population analysis, and as long as the geometry has not converged or the maximum number of geometry cycles has not been reached, GEOPT generates new atomic co-ordinates.

The routine GEOPT consists essentially of three routines, FOCKC, ENGRAD and GEOSTP as can be seen in figure 3. FOCKC calculates the core part of the Fock matrix and overlap matrix, ENGRAD calculates the energy gradients and GEOSTP does a quasi-newton minimisation of the energy by varying the atomic co-ordinates. See for details section 3.2.5.

## 5. Serial performance

In this section we will show our serial timing results for  $\text{Fe}_2(\text{CO})_9$ ,  $\text{Cu}(\text{C}_7\text{H}_6\text{O}_2\text{N})_2$  and

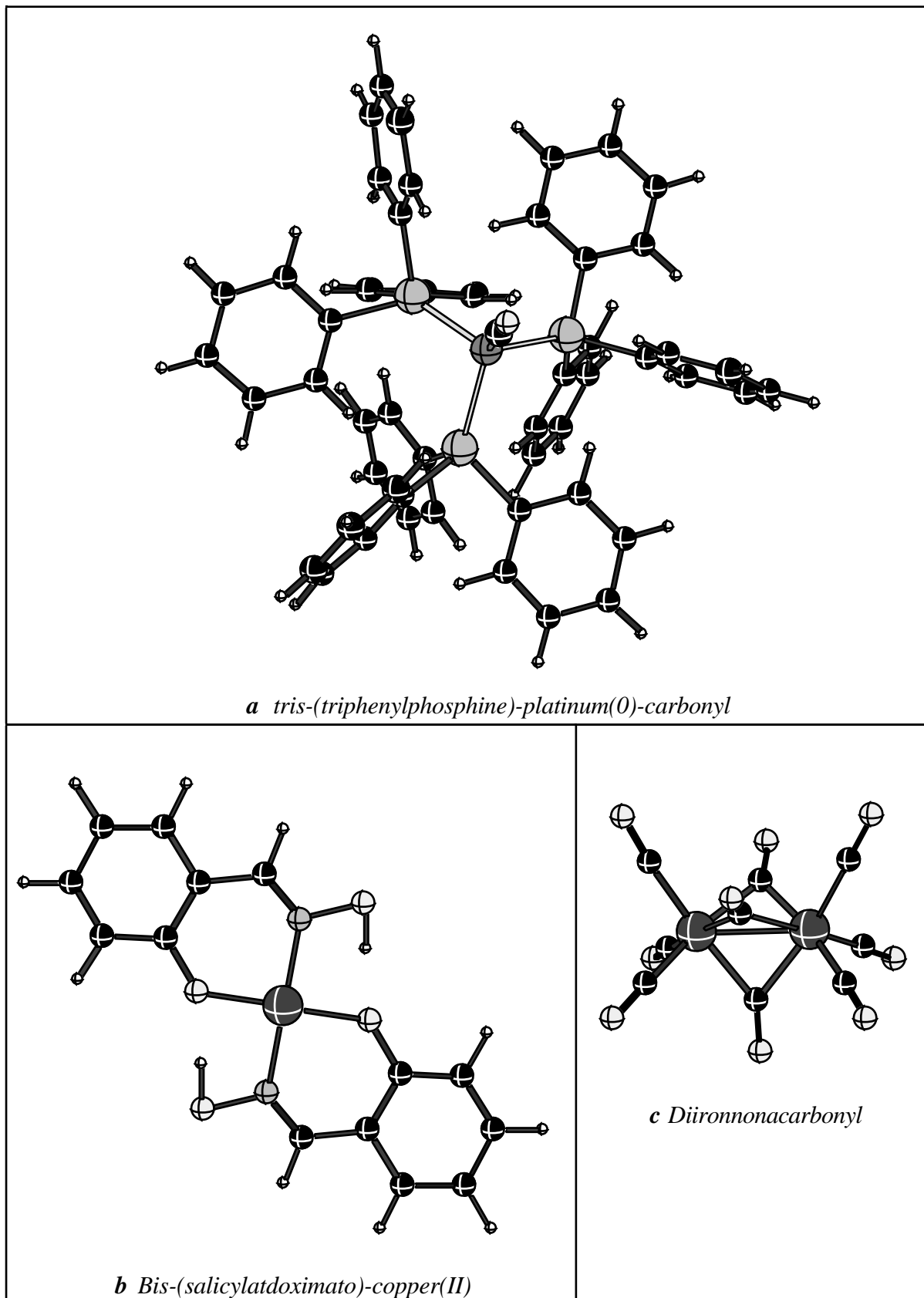
Pt(P(Ph)<sub>3</sub>)<sub>3</sub>CO (see figure 4) on an IBM SP1. These molecules have been used for the benchmarking on the parallel machines. The serial timing results for Pt(P(Ph)<sub>3</sub>)<sub>3</sub>CO were estimated from a two-node run, because we are not able to run this molecule on a single node of the SP1 due to lack of disk space.

In the tables 5.1 to 5.3 the timing results for a single point calculation with and without the gradient corrected density functionals are shown and the geometry optimisation of Fe<sub>2</sub>(CO)<sub>9</sub> with gradient corrections. For most routines mentioned in the previous section the percentage of the overall CPU time is shown.

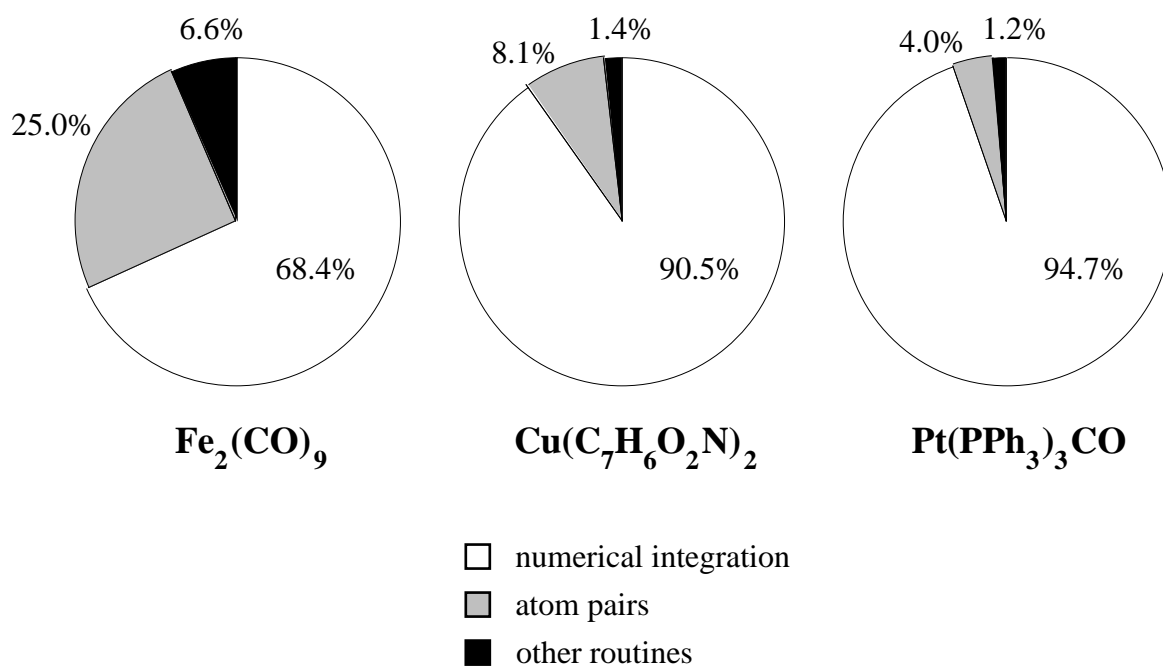
Further information to give some more insight in the size of the calculations: a double- STO basis has been used for all atoms; an 1s frozen core has been used for carbon, oxygen and nitrogen; for copper and phosphor the core has been frozen up to 2p; a 3p frozen core for iron and a 5p for platinum have been used in the calculations; the symmetries used for the different molecules are D<sub>3h</sub> for Fe<sub>2</sub>(CO)<sub>9</sub>, C<sub>1</sub> for Cu(C<sub>7</sub>H<sub>6</sub>O<sub>2</sub>N)<sub>2</sub> and C<sub>1</sub> for Pt(P(Ph)<sub>3</sub>)<sub>3</sub>CO.

From table 5.1 we see that FOCKY is the most expensive routine in all the cases. FITINT can also be expensive when the molecule has symmetry. In figure 5 the serial timing results are represented in a diagram. All the routines that deal with numerical integration have been summed up and all the routines dealing with atom pairs. It shows that the largest amount of time is spent in the routines for the numerical integration. Especially for large molecules with low symmetry the numerical integration becomes even more important. In the second place are the routines dealing with the atom pairs. Thus if we parallelise the routines dealing with numerical integration and atoms pairs we will have most of the program parallelised.

When gradient corrections are added, we see from table 5.2 we see that the portion of the numerical integration only grows, because the gradient corrections which are evaluated by numerical integration are relatively expensive. For the geometry optimisation we see the same in table 5.3. The energy derivatives which are also costly and also evaluated by numerical integration, take a large part of the overall time.



**Figure 4** Molecules used for benchmark runs.



**Figure 5** Diagram of the serial timing results.

**Table 5.1** Serial timing results for single point calculation

Molecule	$\text{Fe}_2(\text{CO})_9$	$\text{Cu}(\text{C}_7\text{H}_6\text{O}_2\text{N})_2$	$\text{Pt}(\text{P}(\text{Ph})_3)_3\text{CO}$
ADF total CPU time	4.67 min	55.5 min	32.3 hours
ORTHON	.5%	.2%	.3%
FITINT**	13.8%	6.5%	3.4%
GENPT*	2.8%	.9%	.2%
PTCOR*	.5%	.3%	.1%
PTBAS*	6.5%	9.4%	20.6%
FOCKY*	50.3%	71.7%	66.0%
SDIIS	.7%	.4%	.3%
DIAGFM	.4%	.2%	.2%
CONPMT**	.3%	< .1%	< .1%
RHOFIH**	2.0%	.7%	.5%
TOTEN*	8.3%	8.2%	7.8%
ESTAT**	8.9%	0.9%	.2%

\*Routine dealing with numerical integration

\*\*Routine dealing with loops over atom pairs

**Table 5.2** Serial timing results for single point calculation with gradient corrections

Molecule	<b>Fe<sub>2</sub>(CO)<sub>9</sub></b>	<b>Cu(C<sub>7</sub>H<sub>6</sub>O<sub>2</sub>N)<sub>2</sub></b>
ADF total CPU time	13.8 min	2.76 hours
PTCRTN	58.6%	59.4%

**Table 5.3** Serial timing results for geometry optimisation with GGA

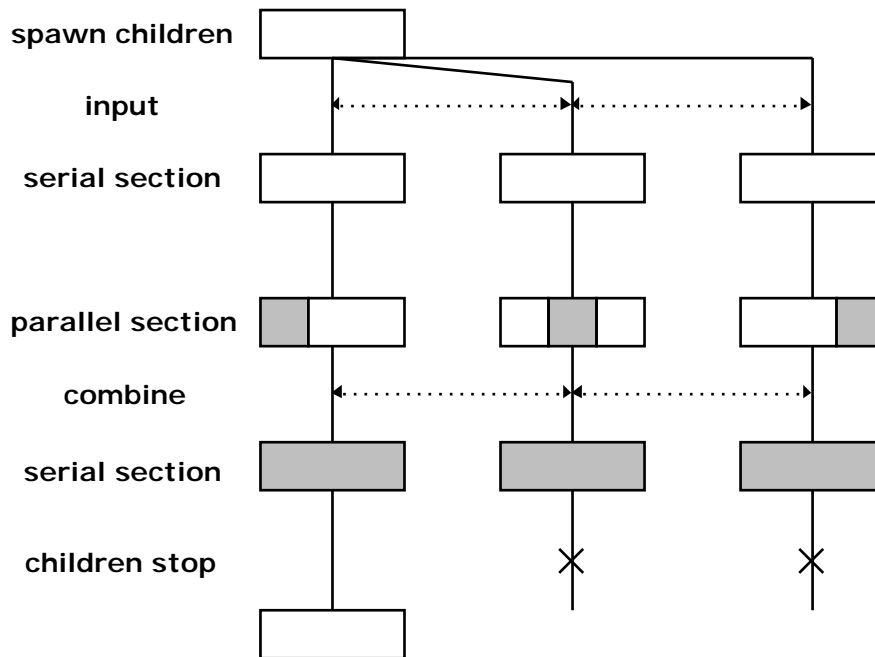
Molecule	<b>Fe<sub>2</sub>(CO)<sub>9</sub></b>
ADF total CPU time	22.1 hours
ATPAIR	.1%
GHOSTP	<< .1%
FOCKC	1.7%
ENGRAD	12.4%

## 6. Parallelisation model

### 6.1. Single Program Multiple Data model

The paradigm of parallelisation that we have chosen for ADF is the single program multiple data (SPMD) model. This model is shown in figure 6. First ADF is started on one of the nodes of the parallel machine. This process is called the parent. Then the parent creates the child processes (normally on different CPU's), reads the input and broadcasts the input to the children. After this on all the nodes we have a copy of ADF running with exactly the same data. So we are using the data replication technique. In the serial parts of the program all these copies perform the same calculation, so they are doing duplicate work. In the parallel parts each copy handles a part of the problem and the results are combined. After all the copies have finished their calculation, the child processes are stopped.

Instead of this model, where all the copies essentially execute the same code, we could have chosen the master-slave model, where only the parent would calculate the serial parts. This would lead to an enormous increase of communication. It costs less time to calculate the small serial parts on all the nodes than to communicate these serial results.



*Figure 6 Single Program Multiple Data*

Another advantage of the SPMD approach is that we can reduce the serial and increase the parallel part of the code step by step during the parallelisation process. As a result we have always a running program. This not only makes debugging much easier, but also the determination of the most time-consuming serial parts that are left over.

Finally, since the structure of the parallel program is virtually identical to the structure of the serial program, it is much easier to maintain both versions simultaneously.

## 6.2. Load Balancing

For the load balancing there are two possibilities: the static and the dynamic load balancing. We expected that communication between the nodes would be expensive and might become a bottleneck on large parallel machines. So we wanted a balancing that would lead to as little communication as possible to get a scalable parallel program. This requirement of small communication time is satisfied by coarse grain static load balancing. The data is kept local and only a very small amount of data is combined over the nodes.

Although dynamic load balancing has the advantage that the parallel program can perform reasonably on loaded machines because of its ability to adapt to external circumstances, we expected that the parallelisation of ADF would benefit more from a static load balancing. The dynamic load balancing did not satisfy our demand of small communication time because the repeated distribution of the data requires much more communication. In the case of dynamic

load balancing each time a node has finished its work, there has to be communication to assign more work to that node, while in the case of the static load balancing this has to be done only once.

The static load balancing in combination with the SPMD model allows us to generate most data local and keep it on file or in memory on the same node throughout the execution of the program. This gives an enormous reduction of disk and memory requirements per node. The resulting files are N-times smaller on an N-processor machine and the size of some matrices are compressed. If the hardware allows it, I/O can be done in parallel which gives a decrease of I/O elapsed time. So in our case parallel machines which have the capability to perform I/O in parallel are preferred.

### **6.3. Communication**

For the communication between the nodes we have used for portability reasons the public domain software library PVM[74], which has been designed to treat a collection of possibly heterogeneous computers as one computer, the so called parallel virtual machine. The PVM system consists of two parts. The first part is a daemon that runs on all the computers forming the virtual machine. The second part provides a library of PVM interface routines for spawning processes and message passing. The processes that are spawned by PVM, such as the copies of ADF on the different nodes, are given a task identifier. In ADF the task identifier is held in MYTID. The routines based on PVM that are used for combining data, like PPCBNR, are discussed in section 7.5.

For the SP2 we have used the vendor implementation PVMe. This enables us to make use of the high speed communication hardware of the SP2.

## **7. Parallelisation of ADF**

### **7.1. Numerical integration**

In the ADF program most of the matrix elements are calculated by using numerical integration as we have seen in section 3. In this section we will discuss the parallelisation of the use of the numerical integration first and then the parallelisation of the generation of the integration points.

#### ***7.1.1. Use of numerical integration***

The numerical integration is parallelised by distributing the integration points over the nodes of the parallel machine. The integration points are divided in NBLOCK blocks, each with

LBLOCK integration points. Then the blocks are distributed over the nodes. This is done by the subroutine PPIBLK, which initialises a function IB2TID. This function returns for each block the task identifier of the node it belongs to. All routines dealing with integration points (FOCKY, PTCRTN etc.) have been parallelised. The balancing of the numerical integration is close to perfect since the amount of work required for each integration point is independent of the integration point considered.

```

do 20 iblock = 1, nblock
  if (ib2tid(iblock) .ne. mytid) goto 20
  read  $\mathbf{r}_k$ 
  for all i
    do 10 k = 1, lblock
      calculate  $\chi_i(\mathbf{r}_k)$ 
10    continue
      write  $\chi_i(\mathbf{r}_k)$  to file
20 continue

```

**Figure 7** Pseudo code for calculation of basis functions in the integration points

In figure 7 pseudo code for the parallelisation of the calculation of the basis functions in the integration points is shown. The loop over the blocks of integration points is the same as in the serial code. One line has been added to the code (namely the second line). If the block is not one of the blocks of this node, the following part is skipped, otherwise the values of all the basis functions in the integration points of that block are calculated. The variable MYTID gives the task identifier of this ADF task. After the loop each node holds on a local file the value of the basis functions in the integration points belonging to that node.

In figure 8 we present pseudo code for the parallelisation of the numerical integration of the elements of the Fock matrix. The loop is again the same as in the serial code. In the first line all matrix elements are set to zero, then the loop over the blocks of integration points is started. Again if the block is not one of the blocks of this node, the following part is skipped, otherwise the partial result is calculated for all the matrix elements. The value of for instance the basis functions that are needed to evaluate the Fock matrix can be obtained from the local files on the nodes. After the loop the incomplete Fock matrices of the nodes are summed by the subroutine PPCBNR.

```

 $F_{ij} = 0$ 
do 20 iblock = 1, nblock
  if (ib2tid(iblock) .ne. mytid) goto 20
  read  $\mathbf{r}_k$ ,  $\chi_i(\mathbf{r}_k)$  etc.
  for all i, j
    do 10 k = 1, lblock
       $F_{ij} = F_{ij} + w_k \chi_i(\mathbf{r}_k) F \chi_j(\mathbf{r}_k)$ 
10    continue
20 continue
ppcblr (F, fsize)

```

**Figure 8** Pseudo code for the calculation of Fock matrix with numerical integration

This approach of parallelisation of the numerical integration, where the data per point is held locally and final partial results are combined, allows the rather time-consuming communication between the nodes to be kept minimal because only small amounts of data are combined.

### 7.1.2. Generation of the integration points

After having parallelised all the routines involving numerical integration, we saw in our timing results that the generation of the integration points (GENPT) takes a significant amount of time. The standard integration scheme of ADF generates the integration points for three different parts of the three-dimensional space; the atomic spheres, the atomic polyhedra, and the layers around the molecule.

Most of the time required by GENPT is consumed by the routines which actually generate the points. The routines generating the points around the atoms (in the atomic spheres and in the atomic polyhedra) have been parallelised by distributing the atoms over the nodes. The routine which generates the integration points in the outer region has been parallelised by distributing the different layers of the outer region. The result is that each node generates only part of the points. For large molecules this load balancing turns out to be satisfactory.

After the generation of the integration points all the nodes have their own set of integration points, possibly not very well balanced. To acquire a good load balancing for the numerical integration some of the points need to be moved to other nodes. To do this each node has to know how many points it will need during the rest of the program. First the routine PPIBLK is called to determine the total number of integration points and to distribute the blocks of integration points in such way that the routines dealing with the integration points have a good load balance. Then the routine PPRDSU is called to determine which node has integration

points in excess and which node needs additional integration points and also to determine the redistribution of the points such that the actual communication is minimal. After this set-up phase, the routine WRINPT loops over all blocks of integration points. If a block of integration points needs to be handled on this node, WRINPT tries to get the points from the list of points generated on this node, but when there are no more points available WRINPT uses PPGATH to get the required points from another node. The complete blocks of points on this node are written to file. After the loop over the blocks has been finished the remaining integration points are made available to the other nodes by calling the routine PPSCAT. At the end of WRINPT, each node holds only those blocks of integration points which it will handle during the rest of the run.

### ***7.1.3. Geometry optimisation***

The geometry optimisation consists essentially of three routines: FOCKC, ENGRAD and GEOSTP. As mentioned in section 5 the execution time of GEOSTP is negligible, so we have not parallelised this routine. The parallelisation of the other routines was rather straightforward: just applying the same techniques as we used in the SCF part for the numerical integration.

The routine FOCKC, which calculates the core part of the Fock and overlap matrices, has the same structure as FOCKY which calculates the valence Fock matrix. It loops over all blocks of integration points, reads relevant data per point (such as core density, core potential), and adds everything together to get the Fock matrix. So the loop over the blocks of integration points in FOCKC has been parallelised in the same way as FOCKY.

ENGRAD calculates the gradient of the energy with respect to the nuclear co-ordinates. It consists of a big loop over all blocks of integration points. For all integration points in that block the relevant contributions to the integrals making up the gradients are calculated and summed up. This routine has also been parallelised by distributing the blocks of integration points and combining the results at the end.

The routine ATPAIR that is called once for the geometry optimisation, has also been parallelised. It calculates atom-atom electrostatic interaction energies that are used by the geometry optimisation algorithm. ATPAIR consists of a loop over all atom pairs. For each atom pair it calls PAIRPT to calculate the electrostatic interaction. PAIRPT loops over a number of atom-atom distances, and for each of these distances the required data is calculated.

PAIRPT has been parallelised by distributing the different distances over all the nodes. The alternative is to distribute the atom pairs, but then it is more difficult to obtain a good load balance. However, since the number of distances is limited (42 in the current implementation)

this routine scales only up to 42 nodes. If the time required by this routine would become excessive, we might parallelise it further by distributing both atom pairs and atom-atom distances.

After each node has calculated the data for their assigned distances, the results are distributed over all nodes using PPGDV.

## 7.2. Atom pairs

In the ADF program we distinguish two kinds of atom pairs; the symmetry-unique atom pairs and the symmetry-equivalent atom pairs. For the density fit only the symmetry-unique atom pairs are used, but in other parts of the program the data of all the atom pairs are needed. In the next part we will first discuss the distribution of only the symmetry-unique pairs for the density fit and in the next part the distribution of all the atom pairs.

### 7.2.1. Symmetry-unique atom pairs

The distribution of the symmetry-unique atom pairs for the routines FITINT and RHOFIH that are concerned with the density fit, and the routine CONPMT which calculates the density matrix on the primitive basis set, is much more involved than the distribution of the integration points. The first difficulty is the determination of the cost of an atom pair. We have on one side the routines FITINT and RHOFIH, and on the other side routine CONPMT. Because FITINT takes much more of the overall execution time than RHOFIH or the other routines, we have determined experimentally the cost of an atom pair in FITINT. From our timing information we obtained that the cost of an atom pair was approximately proportional to the number of integrals  $T_{\mu\nu i}$  (3.1.18) used to calculate the vector  $\mathbf{t}$  (3.1.17) in RHOFIH.

The routine PPPAIR, which does the load balancing for the atom pairs, distributes the atom pairs by first sorting the atom pairs according to decreasing number of integrals  $T_{\mu\nu i}$  and then assigning the next atom pair to the node with the lowest cumulative weight. The function IP2TID is initialised by PPPAIR. It has two arguments: the first is for the atom pairs that are used and the second for the number of the atom pair. When the first argument is equal to 1, the function returns for the symmetry-unique atom pair the task identifier of the node it belongs to and for each symmetry-equivalent atom pair it returns the value of a non existing node.

The load balancing achieved by this distribution is perfect for FITINT, at least if there are enough atom pairs to be distributed. It also works very well for RHOFIH, because in RHOFIH the most time-consuming part is the calculation of the vector  $\mathbf{t}$  (3.1.17) and this is of course

proportional to the number of integrals  $T_{\mu\nu}$  .

```
do 20 ipair = 1, npair
  if (ip2tid(1, ipair) .ne. mytid) goto 20
  Calculate fit integrals for ipair
  Write to file
20 continue
```

```
do 20 ipair = 1, npair
  if (ip2tid(1, ipair) .ne. mytid) goto 20
  Read from file
  Fit density for ipair
20 continue
ppcbr (fit coefs)
```

*Figure 9 Pseudo code of FITINT and RHOFIH*

In figure 9 pseudo code for FITINT and RHOFIH is presented. Here we see again the advantage of the static load balancing, as we saw in the case of the numerical integration. The routine that does the preparation, in this case FITINT, writes the data to a local file on the node, and the routine which actually does the calculation, in this case RHOFIH, reads the data from the local file and calculates the fit coefficients. Then only a relatively small amount of data, the fit coefficients, have to be combined at the end of RHOFIH.

The routine CONPMT has the same distribution of the atom pairs as RHOFIH and FITINT. We can see from equation (3.1.17) that only the elements of the density matrix that belong to the atom pairs on a node are needed on that particular node. Also the density of the symmetry equivalent atom pairs is not fitted by RHOFIH, so the elements of the density matrix of those atom pairs are not needed and therefore not calculated.

```
do 20 ipair = 1, npair
  if (ip2tid(1, ipair) .ne. mytid) goto 20
  for all  $\mu\nu$  of ipair
    Calculate  $P_{\mu\nu}$ 
20 continue
```

*Figure 10 Pseudo code for the construction of the density matrix*

In figure 10 pseudo code for the calculation of the density matrix is represented. Again we benefit from the choice of the static load balancing. The elements of the density matrix that are calculated on a particular node by CONPMT are also the elements of the density matrix that are needed in RHOFIH to fit the density as we can see from equation (3.1.17). No communication is needed in this case. This static distribution gives us also another advantage. While a certain node does not need the matrix elements of the density matrix that are used by other nodes, we can compress the density matrix on a node by taking as the size of this matrix all the elements that belong to the atom pairs of that particular node. So we also achieve a compression of the needed memory that scales with the number of nodes.

In the routine CONPMT the cost of an atom pair is proportional to the square of the number of primitive basis functions of that atom pair. It is not obvious that the chosen balancing, which is based on the balancing for FITINT, would be proper for this routine, but our timing results show that the CPU times are well balanced. This approximation would be justified if the number of fit functions would be equal for all the atom pairs. Then CONPMT would be balanced as well as FITINT. Our approximation is further justified by the fact that CONPMT does not take much computational time.

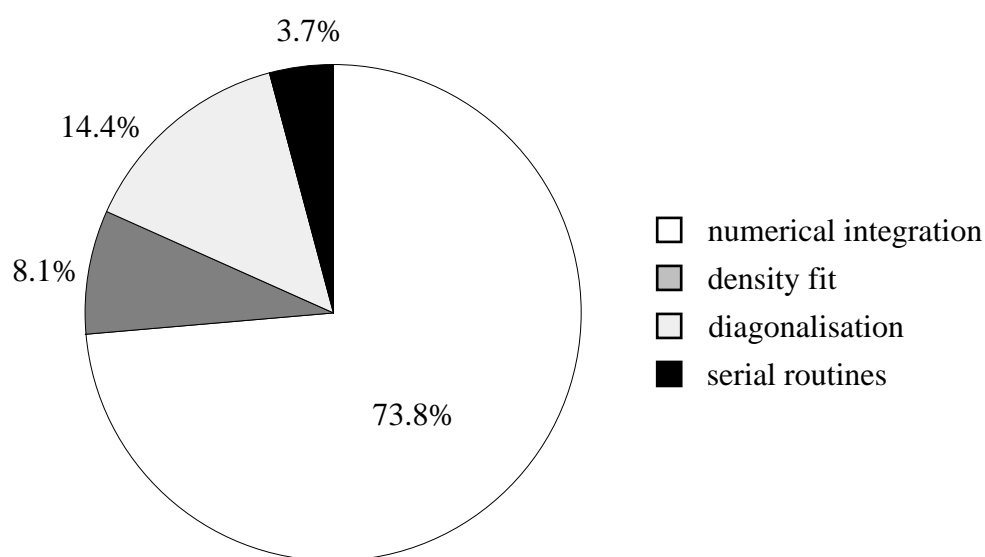
### ***7.2.2. All atom pairs***

In the ADF program there are some routines with loops over the atom pairs that do not use symmetry. For instance CLSMAT, which calculates the overlap matrix on the primitive basis set, and CNTPMT which calculates for the total density matrix on the primitive basis set, which is needed for the population analysis. For these routines the cost of an atom pair is proportional to the number of elements of the overlap or density matrix. The distribution of the atom pairs is done by PPPAIR by first sorting the atom pairs according to decreasing number of elements of the density matrix and then assigning the next atom pair to the node with the lowest cumulative weight. The function IP2TID returns with the first argument set to 2 for each atom pair the task identifier of the node it belongs to. All the atom pairs are distributed over existing nodes.

The loop for constructing the density matrix in the routine CNTPMT looks almost the same as the loop in CONPMT for constructing the partial density matrix. The only difference in that loop is that IP2TID (1,IPAIR) is replaced by IP2TID (2,IPAIR) to handle all the atom pairs and the density matrix of the different nodes is combined at the end of the loop.

### 7.3. Diagonalisation

After we had parallelised all the routines concerning the numerical integration and the atom pairs, we were confronted with the diagonalisation bottleneck, especially for the larger molecules without symmetry. In figure 11 our timing results for  $\text{Pt}(\text{P}(\text{Ph})_3)_3\text{CO}$  on an eight node IBM SP1 with a preliminary parallel version of our program are shown. It clearly shows that the diagonalisation has to be handled in order to be able to get linear speedups.



**Total CPU time: 13.9 hours**

*Figure 11 Timings on a 8 node IBM SP1 with a preliminary parallel version of ADF*

It was therefore necessary to obtain a parallel diagonalisation algorithm for real symmetric matrices. To solve this problem we applied the following approach. Our new diagonalisation routine is based on a tridiagonalisation followed by the determination of the eigenvalues and eigenvectors and a transformation of the eigenvectors back to the primitive basis. For the tridiagonalisation we use a serial LAPACK[75] routine (DSYTRD) and for the determination of the eigenvalues and eigenvectors we also make use of serial LAPACK routines. These routines are capable of handling only a subset of the eigenvalues and eigenvectors: either by index range or by energy window. We use the routine DSTEBZ to determine the eigenvectors by bisection, and then we use the routine DSTEIN to determine the eigenvectors by inverse iteration. For the transformation of the eigenvectors to the original basis we use the routine DORMTR.

In the parallel version of our diagonalisation routine each processor handles only the eigenvalues and eigenvectors within a certain index range, with the ranges chosen to balance

the computations over all processors. We have taken considerable care not to split degenerate eigenvalues over the different processors since that does not lead (in this approach) to a complete orthonormal set of eigenvectors. When all processors have finished the resulting eigenvalues and eigenvectors are combined over all processors, so at the end each processor holds all eigenvalues and all eigenvectors, just as the original serial code. The new diagonalisation routines still contains a serial part, namely the tridiagonalisation step. We intend to replace this as soon as it becomes a bottleneck.

As it was easy to implement and it would accelerate the serial and parallel run the diagonalisation routine determines only the occupied eigenvalues and eigenvectors during the SCF. The unoccupied eigenvectors are not needed for the determination of the new density matrix. Only in the last iteration the unoccupied eigenvalues and eigenvectors are determined because they might be needed in the following parts of the program.

#### **7.4. DIIS**

It turned out that the SDIIS routine was one of the serial bottlenecks in the SCF part. So this routine also had to be parallelised.

The DIIS procedure requires an error vector which is defined in ADF as the commutator of the Fock matrix and the density matrix. We have distributed the elements of the error vector over all nodes, since this leads to the parallel computation of the error vectors of each cycle, as well as the parallel computation of the overlap matrix of the error vectors, and finally it leads to the distributed storage of the error vectors and Fock matrices.

The calculation of the commutator has been parallelised by modifying a matrix multiply routine (PPMMPP) such that only the elements of the error vector required on this node are calculated. The method scales well with the number of processors, and no communication is involved since both the current Fock and the current density matrix on the orthogonalised SFO's are available completely on each node.

The calculation of the overlap matrix is simple then. The old error vectors and the old elements of the overlap matrix are read from file and the dot product of the new error vector with each of these old error vectors is calculated to generate the new elements. Since the error vectors have been distributed, only the local part of the error vectors is written to and read from file, thus parallelising the I/O operations. This dot product leads to a partial overlap matrix on each node, and the final overlap matrix is generated by combining the partial overlap matrices over all nodes. Since this is just a small matrix (normally less than 10 error vectors are involved) this is

not a bottleneck.

After solving the linear equations the new Fock matrix is determined as a linear combination of the old Fock matrices. We have stored these Fock matrices partially by distributing the elements just as we have distributed the elements of the error matrices. When creating the actual linear combination, each node creates a partial linear combination. The result is a distributed Fock matrix which needs to be gathered to make it available completely on all nodes. This gather is done using PPGDV (see section 7.5.2). This method leads also to a code which runs completely parallel: both the CPU intensive parts (the calculation of the commutator and the linear combination of the matrices) and the I/O intensive part.

### **7.5. PP Library**

PVM is a low level library, so we decided to set up a PP Library for ADF which contains high level routines. It contains e.g. routines for combining partial results over the nodes. These routines are based on the PVM message passing routines. This high-level library enables us to hide the PVM calls in a few routines and we can therefore easily use another message passing library for our program by changing only those routines instead of going through the whole program.

Besides the routines that combine incomplete or distributed matrices, the PP Library also contains routines that find the maximum or minimum value of a variable over the nodes. Other utilities are for instance the barrier routine PPBARR, which synchronises all the nodes by having all the children send an arbitrary message to the parent. Next the parent broadcasts an arbitrary message to the children. PPBARR is particularly useful to be called when the children know who their parent is but do not know of the existence of their siblings. It can also be used before a timer call. In this way we are able to measure the pure elapsed time of the individual routines without time spent in the other routines.

The library also contains a routine PPINIT which initialises the parallel version of ADF. This routine is called by the parent and the children. The parent reads the first line of input, and based upon that line generates a number of children. After creating the children, PPINIT waits in a barrier so after the call to PPINIT one can assume that the children are alive and well. All tasks will be added to the PVM group 'ADF'.

For the calculations on one of the platforms (SP2) we used the IBM implementation of PVM (PVMe). It enables to utilise the high speed communication hardware of the SP2. However the calls in PPINIT had to be adapted to make use of PVMe due to incomplete compatibility with

the public domain version of PVM.

The library contains high-level communication routines for combining incomplete or distributed matrices. For these routines a number of algorithms are available. At installation time the user can decide the algorithm to use. We have defined the incomplete matrices as those that have the same size as in a serial run, but the matrices on the different nodes contain only a part of the total result. So the local matrices have to be summed over the nodes to get the final matrix. The distributed matrices on the other hand have their elements distributed over the nodes. Their size per node will be smaller than in a serial run. To get the total matrix the distributed elements have to be gathered.

### ***7.5.1. Combining incomplete matrices***

For the routine that handles the combining of the incomplete matrices, PPCBNR, a choice between three different algorithms [76,77] can be made.

The first algorithm is the well-known binary tree algorithm, see figure 12a. In this algorithm each even node gets the incomplete matrix from the uneven node on the right and adds these two incomplete matrices. In the next step the uneven nodes are left out and the process is repeated for the remaining nodes. This continues until the total matrix is on the parent. Then the total matrix is broadcast to the children by walking up the binary tree.

In figure 12b the "double binary tree" algorithm is presented. In the first step each pair of nodes next to each other exchange their incomplete matrix and sum these up. In the next step this is done again but now between the pair of nodes that are one node further away from each other. This exchanging and summing is continued until all the nodes contain the total matrix.

In double binary tree algorithm all the nodes send and receive at the same time and they also sum the incomplete matrices at the same time. The advantage of this algorithm is that less communication is required. In the normal binary tree the number of communication steps is equal to  $2\log N$  ( $N$  = number of nodes). But with the double binary tree the number of communication steps is only equal  $\log N$ .

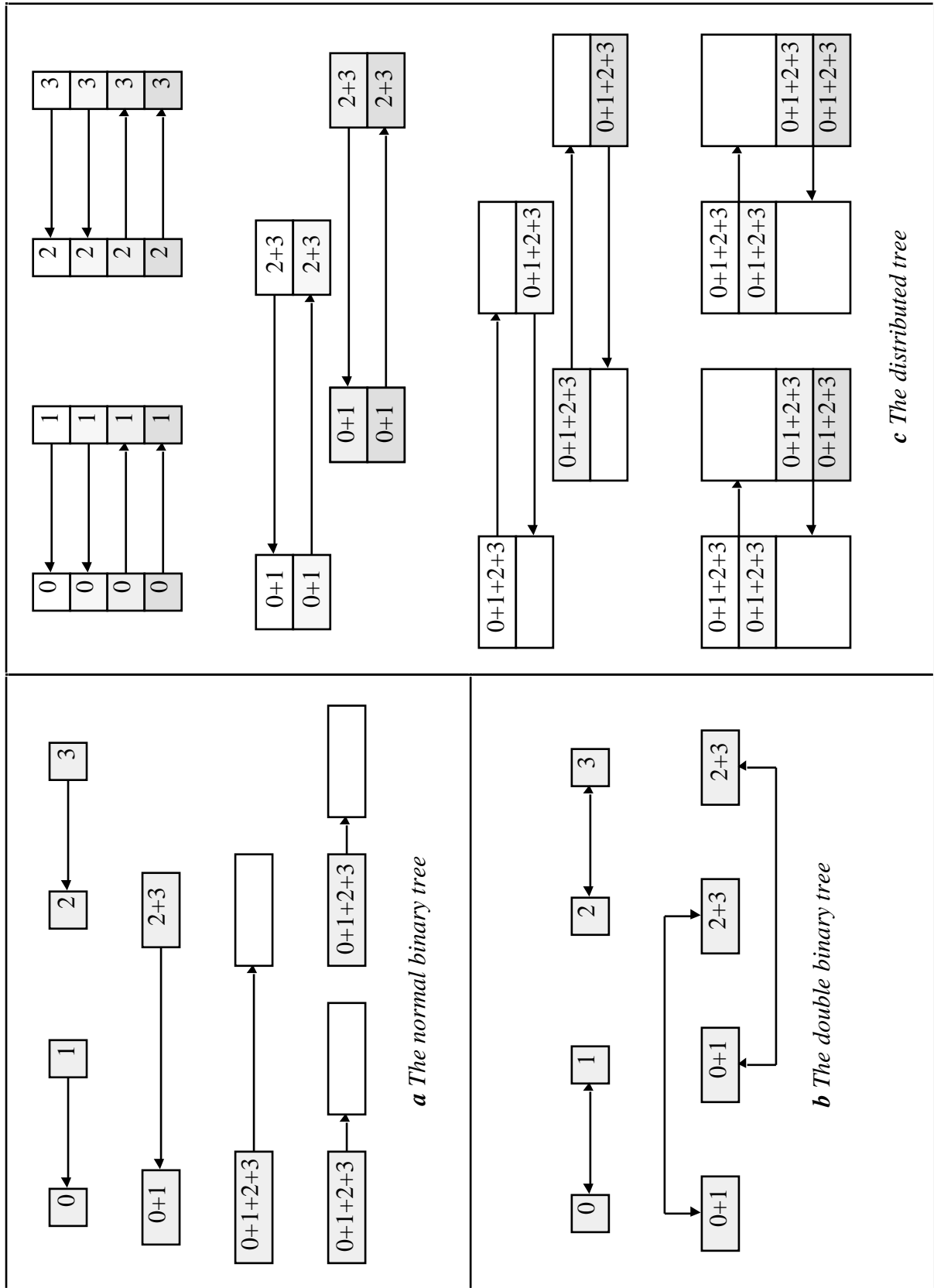


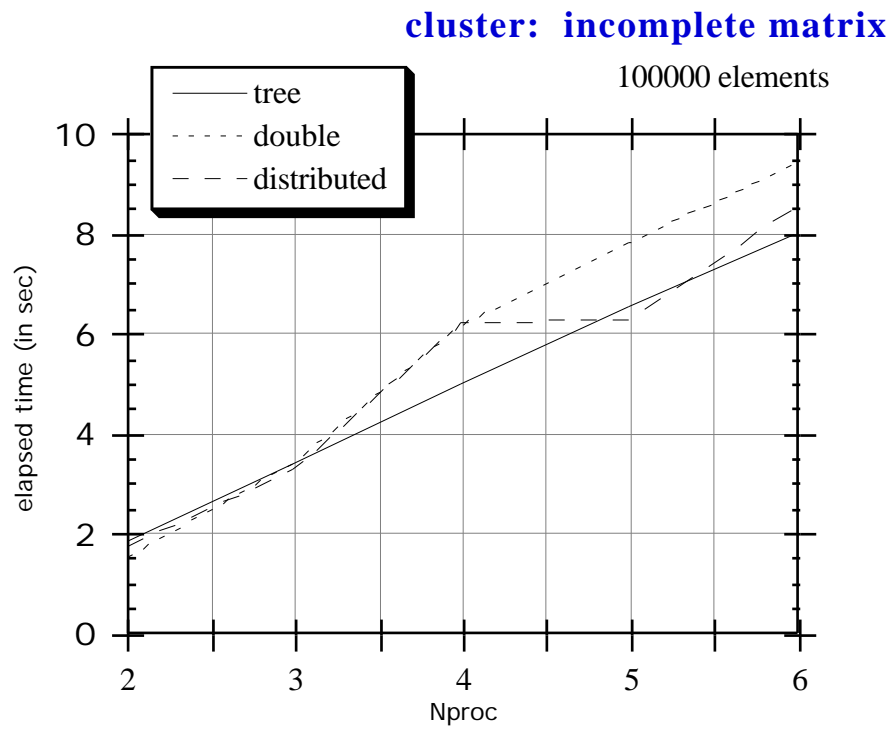
Figure 12 Algorithms for combining incomplete matrices residing on all nodes.

The last algorithm, the "distributed tree" algorithm is shown in figure 12c. This algorithm does the summation of the matrix elements in parallel. Each pair of nodes next to each other exchange half of their incomplete matrix and sum this up. In the next step the pair of nodes that are one node further away from each other repeat this process. The half of the incomplete matrix that they contain, is divided in two parts. One quarter of the incomplete matrix is sent to the other node and the quarter that remains on the node, is summed up with the part received from the other node. The process is repeated until each node contains a part of the total matrix. So the total matrix is now distributed over the nodes. Each part of the total matrix on a node is equal to the number of elements of the total matrix divided by the number of nodes. To get the total matrix on all the nodes, each node exchanges its part with the node that has been the last one with which it has communicated. This process is repeated until finally half of the total matrix is on each node and is exchanged between the two neighbour nodes.

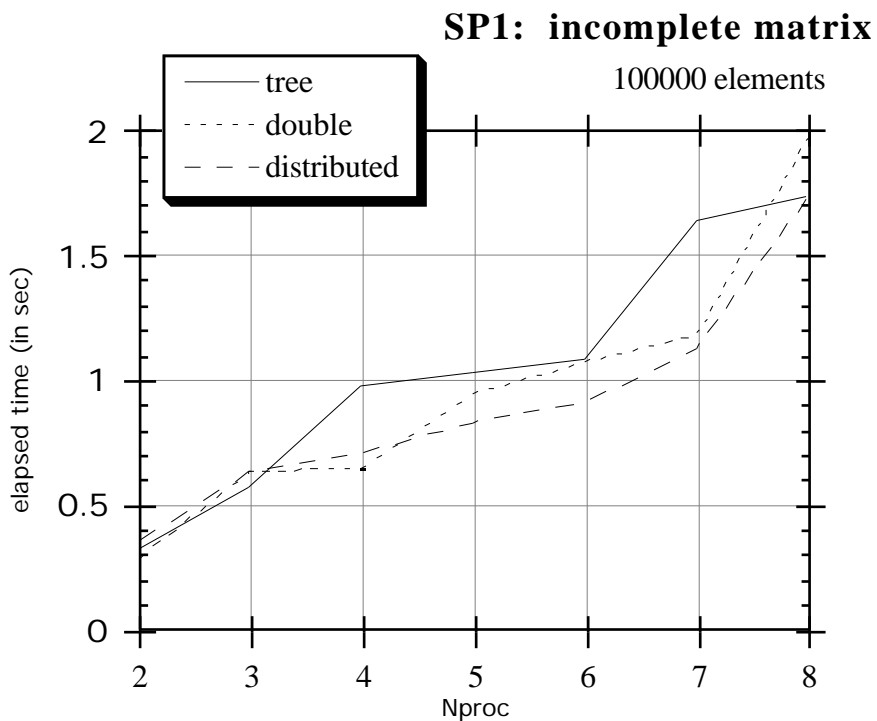
This algorithm is particular suitable when the summation of the matrix becomes expensive, but the communication is relatively cheap. In the double tree algorithm each node did the same summation but in the distributed tree algorithm the summation is done in parallel. The number of communication steps is equal to  $2\log N$ .

For simplicity we have assumed that the number of nodes equals a power of 2. For the binary tree this is not a problem. The other two algorithms are only used for the largest power of two partition of the nodes. The data from the nodes in excess are explicitly handled before and after the combine requiring two additional steps.

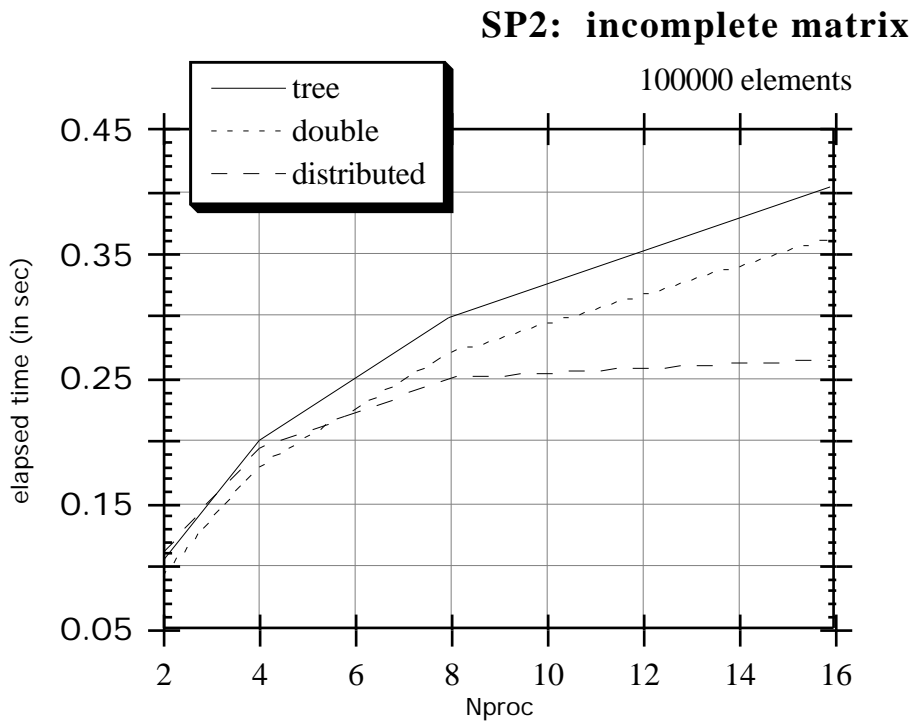
To choose the algorithm to be used on the parallel platform, we have measured the three different algorithms on the machines that are used. The platforms used are a workstation cluster of 6 IBM RS6000/250 workstations connected by ethernet, an 8-node IBM SP1 with FDDI for communication and the IBM SP2 with the high performance switch connecting the processors. The three algorithms were used to combine a matrix of 1000 elements and a matrix of 100000 elements. In figure 13, 14 and 15 the results for the combining of the largest matrix are given for the workstation cluster, the SP1 and the SP2. It shows that the fastest algorithm on the cluster is the binary tree. For the SP1 we also choose the binary tree. On the SP2 however the distributed tree is much faster than the other two so PPCBNR uses the distributed tree algorithm on the SP2. The results for the smaller matrix give the same conclusions.



*Figure 13 Comparison of algorithms for combining on a workstation cluster*



*Figure 14 Comparison of algorithms for combining on a 8-node IBM SP1*

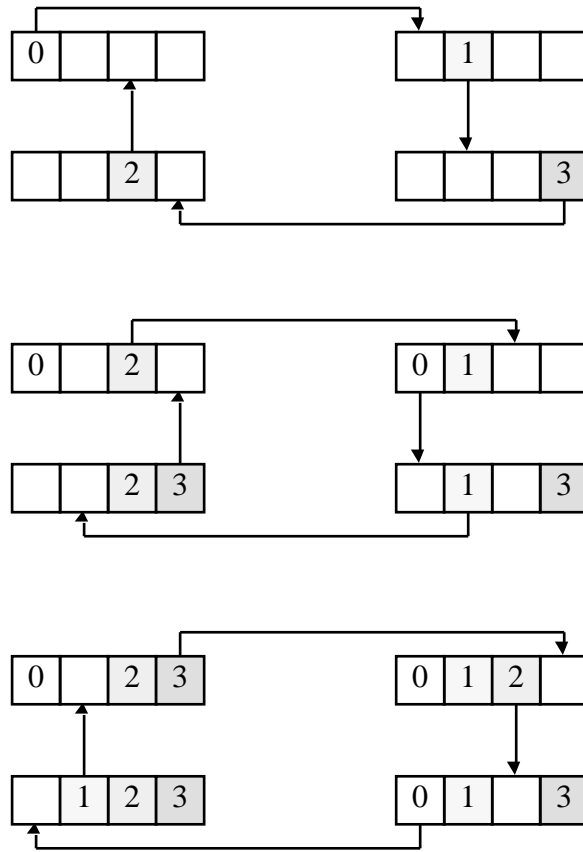


*Figure 15 Comparison of algorithms for combining on a 16-node IBM SP2*

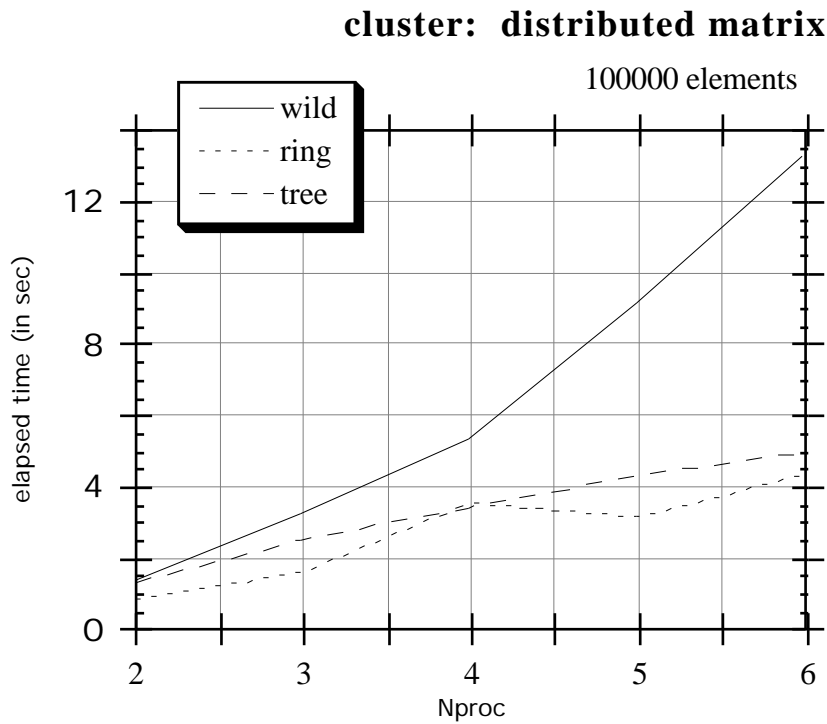
### 7.5.2. Gathering distributed matrices

The combining of the distributed matrices is done by the routine PPGDV. It has the choice between three different algorithms. The first algorithm, the "wild" algorithm is rather trivial. Each node just sends its part of the matrix to the other nodes. The second algorithm is the so-called ring algorithm. In figure 16 this algorithm is shown. The nodes of the parallel machine are seen as a ring. In each step a node sends to its right node the part of the matrix that the receiving node does not contain and receives a new part of the matrix from its left node. This process is repeated until the whole matrix is on all the nodes. The last algorithm is the binary tree algorithm, see figure 12a. The difference with binary tree of the incomplete matrices, is that no summation is needed and the indices have to be sent.

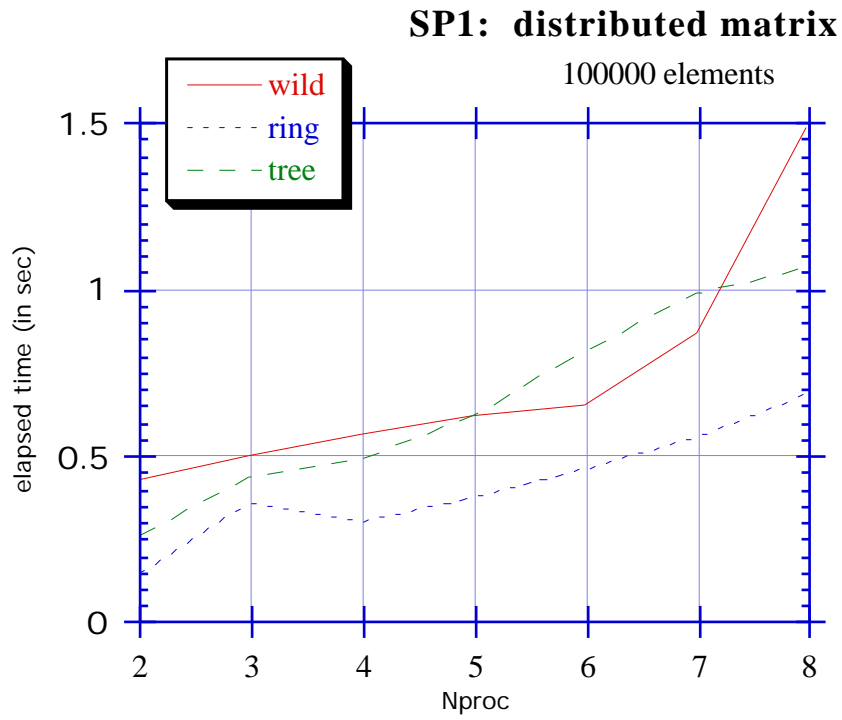
We have measured on the different platforms the time for gathering distributed matrices of 1000 and 100000 elements. The results for gathering the largest matrix are shown in figures 17, 18 and 19. From these figures we can immediately conclude that the ring algorithm is the fastest on all platforms. The same result was obtained for smaller matrices.



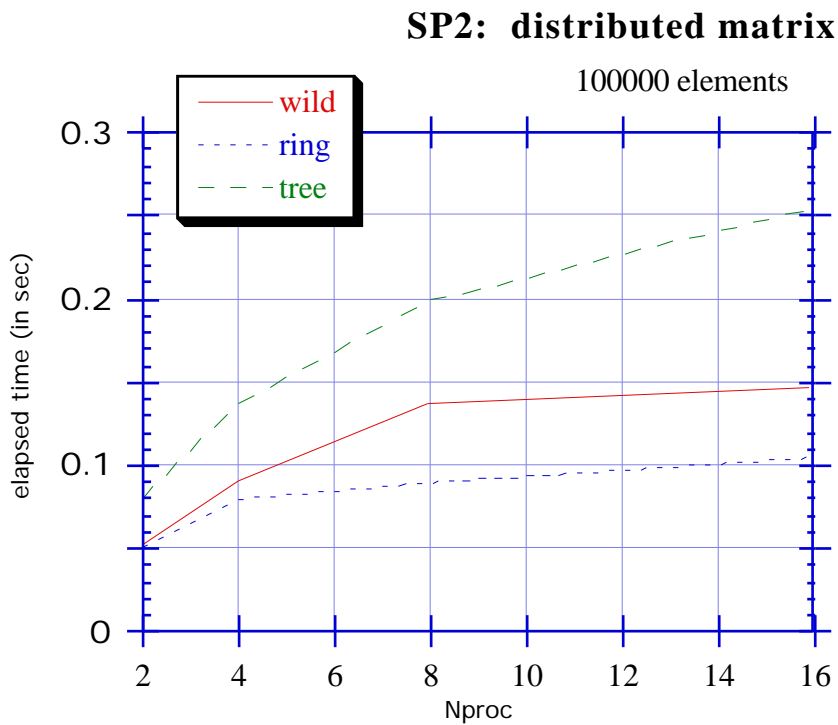
**Figure 16** The ring algorithm for gathering distributed matrices



**Figure 17** Comparison of algorithms for gathering on a workstation cluster



*Figure 18 Comparison of algorithms for gathering on a 8-node IBM SP1*



*Figure 19 Comparison of algorithms for gathering on a 16-node IBM SP2*

## 8. Parallel Performance

To show the performance of ADF running on parallel machines we have chosen molecules that will illustrate the different aspects of ADF. A single-point calculation for a large molecule, Pt(P(Ph)<sub>3</sub>)<sub>3</sub>CO; a single-point calculation with non-local corrections for a medium-sized molecule, Cu(C<sub>7</sub>H<sub>6</sub>O<sub>2</sub>N)<sub>2</sub> and a geometry optimisation with non-local corrections was done for a medium-sized molecule, Fe<sub>2</sub>(CO)<sub>9</sub>. The basis set and frozen cores used, are the same as in section 5.

The calculations have been done on different hardware platforms: an 8-node IBM SP1 using FDDI network for communication, a 28-node IBM SP2 using a high-performance switch for communication, and a workstation cluster of 6 IBM RS6000/250 workstations connected by ethernet.

### 8.1. Timing definitions

For the presentation of our timing results we have used the following definition for the speedup:  $S = T_s / T_p$ , where  $T_s$  and  $T_p$  are the elapsed times of the whole program executed on a single node respectively a number of nodes. In the elapsed time the startup time of all the children is also included, because that is the real time that a user has to wait for his job to be finished.

To find out if communication, load imbalances, I/O or page faults influence the measured speedup we compare our speedup results with the speedups from Amdahl's law[78]:

$$T_p = t_s + \frac{t_p}{n} \quad (8.1.1)$$

where  $t_s$  and  $t_p$  are the elapsed time of the serial respectively parallel part of the program executed on a sequential computer. There are two main reasons for an increase of the deviation between the curve of the measured speedup and the speedup obtained from Amdahl's law. It can be caused by an increase of the communication time between the nodes or by load imbalances. To obtain the times  $t_s$  and  $t_p$  the program prints the percentage of time that it runs in the parallel parts, when executed on a single node. When the run is on more nodes, it estimates this percentage from Amdahl's law by assuming Amdahl's law to be exact.

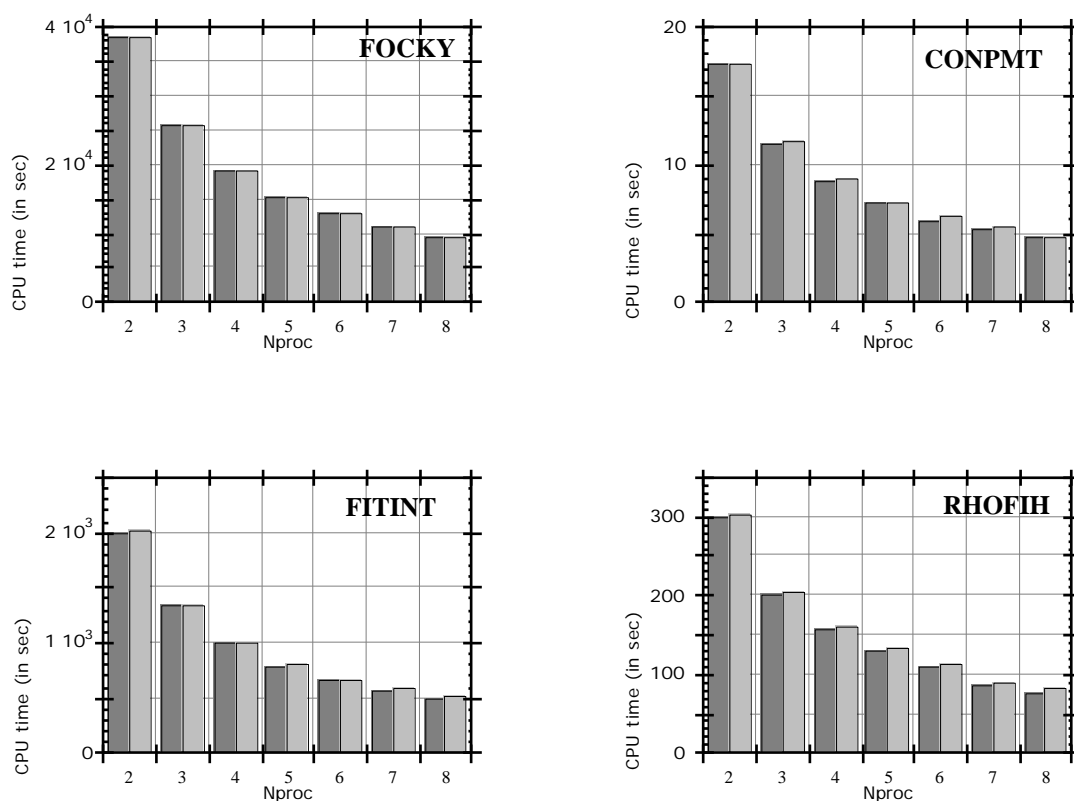
Further for the timing of some individual routines we have used PPBARR to synchronise all the tasks before starting timers for that routine. This prevents imbalances from previous routines to be measured as part of the routine measured. The speedup shown for these individual routines have all been measured on the 8-node IBM SP1 using FDDI.

In some of the graphs where two speedups are shown, one of the speedups is raised by one otherwise only one line would be visible. In the graphs with three speedups one of them is raised by two.

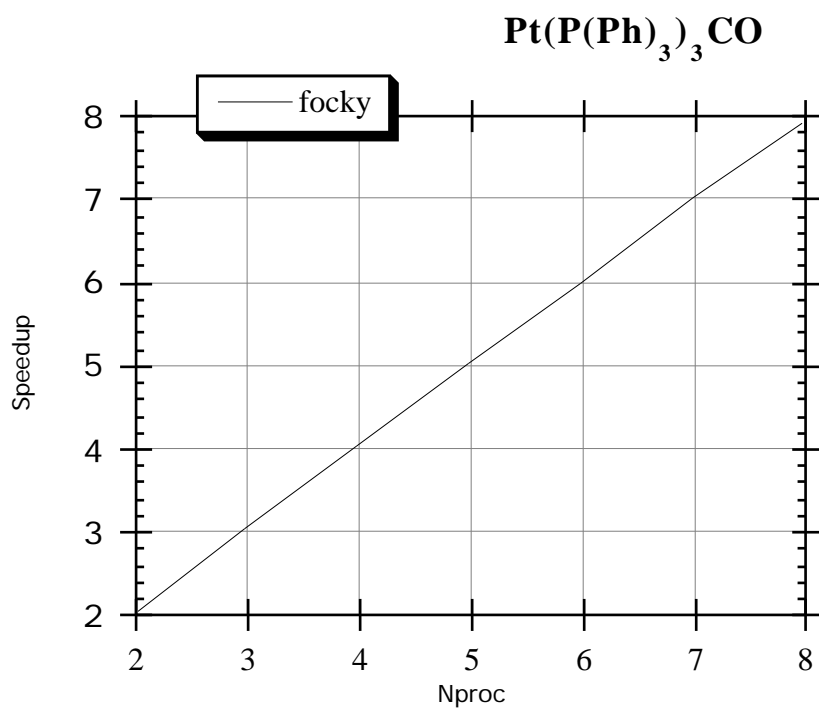
## 8.2. Single point: Pt(P(Ph)<sub>3</sub>)<sub>3</sub>CO

In this section the results for the parallelisation of a single point calculation are shown. The molecule considered here is Pt(P(Ph)<sub>3</sub>)<sub>3</sub>CO. As mentioned before this molecule cannot be run on a single node, therefore the speedup at 2 nodes is set to 2.

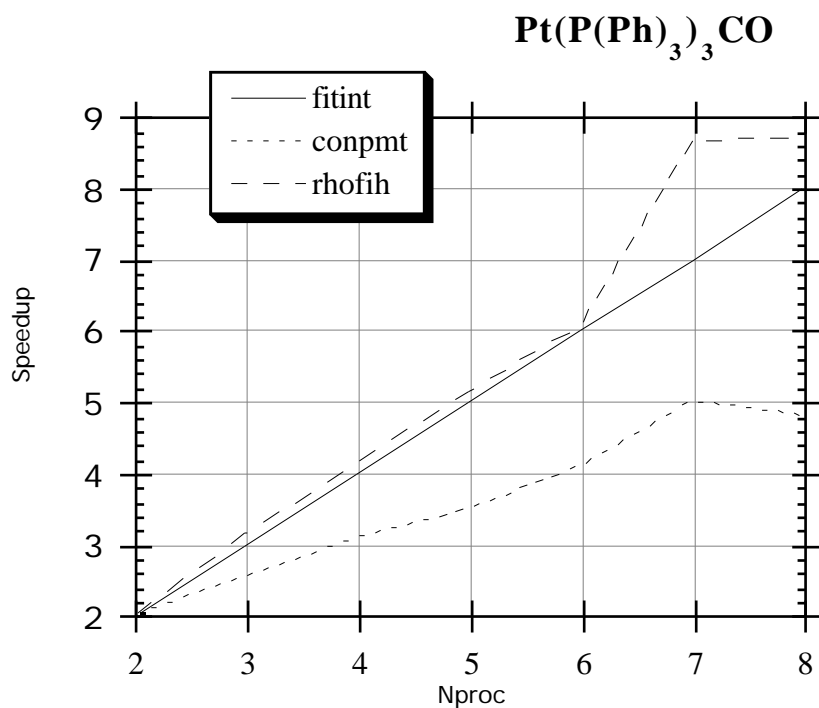
Figure 21 shows to the routine FOCKY that calculates the Fock matrix by numerical integration. As we expected this routine scales exactly with the number of nodes because of the perfect load balancing that is achieved with the numerical integration. This can be seen in figure 20 where the CPU times for the fastest and slowest node are shown.



**Figure 20** Load balancing for the routines FOCKY, FITINT, RHOFIH and CONPMT.



**Figure 21** Speedup for the routine *FOCKY* on the SP1.



**Figure 22** Speedup for the routines *FITINT*, *CONPMT* and *RHOFIH* on the SP1.

The speedups for the routines that are concerned with the density fit are shown in figure 22 (see also table 8.1). We see that the routine FITINT which imposes the load balancing for the atom pairs, scales exactly with the number of processors. This is due to the fact that for this large molecule without symmetry there are enough atom pairs to distribute over the nodes and that FITINT does not communicate. Therefore none of the nodes will be waiting for the others to finish.

The routine RHOFIH scales satisfactory, but CONPMT does not scale so well. The speedup of the routine CONPMT is not so good due to 10% of serial time left for the 2-node run. However this is not so important because it less than 1% of the overall elapsed time.

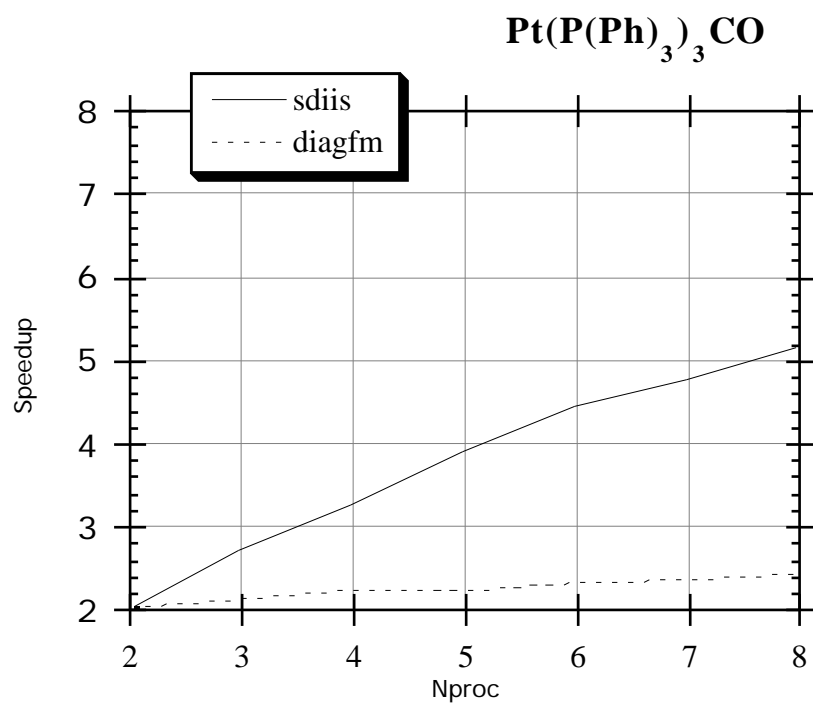
The graph shows further that the speedup of RHOFIH increases dramatically at 7 nodes to 8.6 and that of CONPMT peaks at 7 nodes. This behaviour is caused by an enormous decrease of the number of page faults when going form 6 to 7 nodes.

To examine the load balancing for the routines that deal with atom pairs the CPU times for the fastest and slowest node are shown in figure 220. We can see that the chosen load balancing works ideally for FITINT and RHOFIH but also for CONPMT it seems to be perfect.

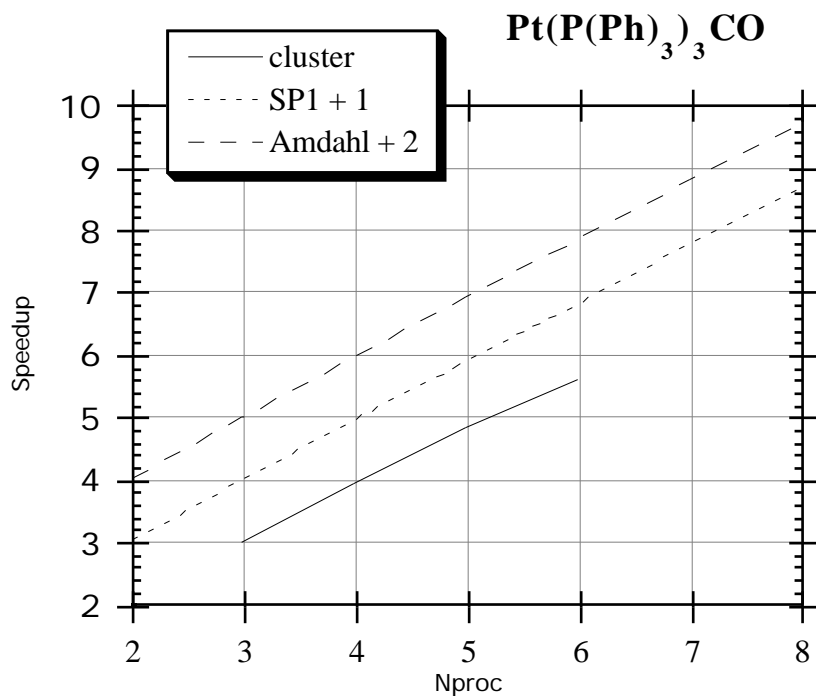
In figure 23 the speedups for the routines SDIIS and DIAGFM are represented (see also table 8.1). The routine SDIIS scales reasonably with the number of processors. A few percentages of serial time make Amdahl's law show up. The speedup of the routine DIAGFM is poor. In the future when the diagonalisation becomes a bottleneck on a larger parallel machine, we need to handle it in another way.

**Table 8.1** Elapsed time of routines on 8 nodes of the IBM SP1

<b>Routine</b>	<b>Speedup</b>	<b>Elapsed time (in sec)</b>
FOCKY	7.9	9918
FITINT	8.0	555
CONPMT	4.8	8
RHOFIH	8.7	494
SDIIS	5.1	91
DIAGFM	2.4	131



**Figure 23** Speedup for the routines SDIIS and DIAGFM on the SP1.



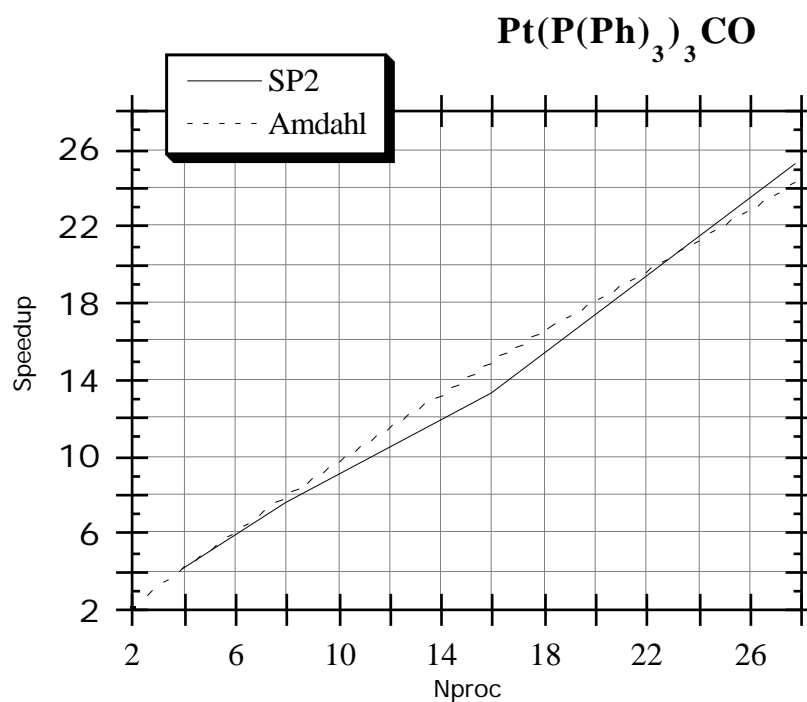
**Figure 24** The speedups for the cluster and the SP1 and the speedup by Amdahl's law.

**Table 8.2** Elapsed time of ADF program for Pt(P(Ph)<sub>3</sub>)<sub>3</sub>CO

Parallel platform	Nproc	Speedup	Elapsed time (in sec)
cluster with ethernet	6	5.6	48891
SP1 with FDDI	8	7.6	16545
SP2 with switch	28	25.4	3668

The last two figures of this section show the speedups for the whole program on different platforms and the speedup predicted by Amdahl's law (see also table 8.2). On a single node 99.4% of the elapsed time is spent in the parallel parts. We see that the speedup of the whole program does not deviate from the speedup predicted by Amdahl's law. For the number of nodes used there is no real communication problem. The ethernet connection that is used for the cluster is quite acceptable for 6 workstations.

Figure 25 shows that on the SP2 we get at 28 nodes a super linear speedup. The speedup exceeds the speedup predicted by Amdahl's law. This is caused by a reduction of the number of page faults. Going from 8 to 16 nodes the number of page faults is halved, but from 16 to 28 nodes the number of page faults decreases by a factor of 10.



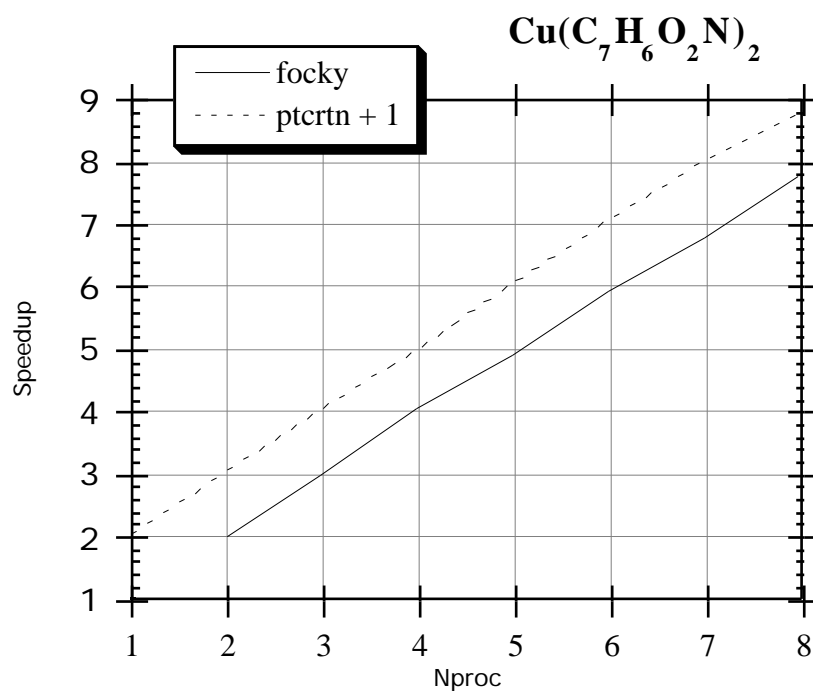
**Figure 25** The speedup for the SP2 and the speedup by Amdahl's law.

### 8.3. Gradient corrections: $\text{Cu}(\text{C}_7\text{H}_6\text{O}_2\text{N})_2$

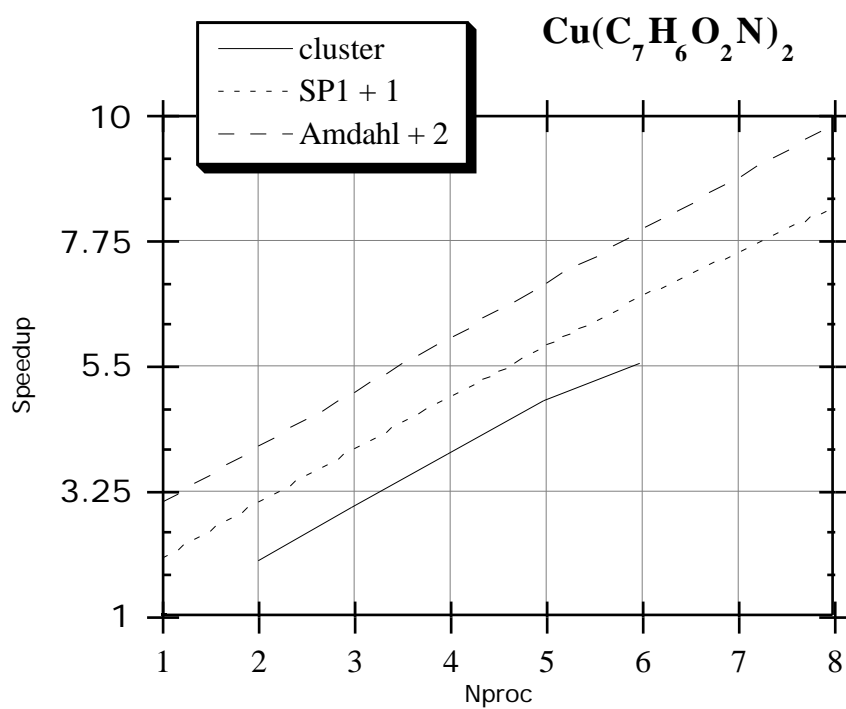
The results for the single point calculation with gradient corrections during the SCF do not differ much from the previous results. The overall elapsed time is again dominated by the numerical integration. Figure 26 shows the speedup for the routines FOCKY and PTCRTN. The last routine calculates the gradient corrections to the exchange-correlation potential in the integration points. The difference between the two routines is that FOCKY has to combine the Fock matrix at the end while PTCRTN does not communicate at all. As we expected both routines scale exactly with the number of nodes (see table 8.3). Further we see that for the number of nodes used, the communication in FOCKY is not visible yet.

**Table 8.3** Elapsed time of routines on 8 nodes of the IBM SP1

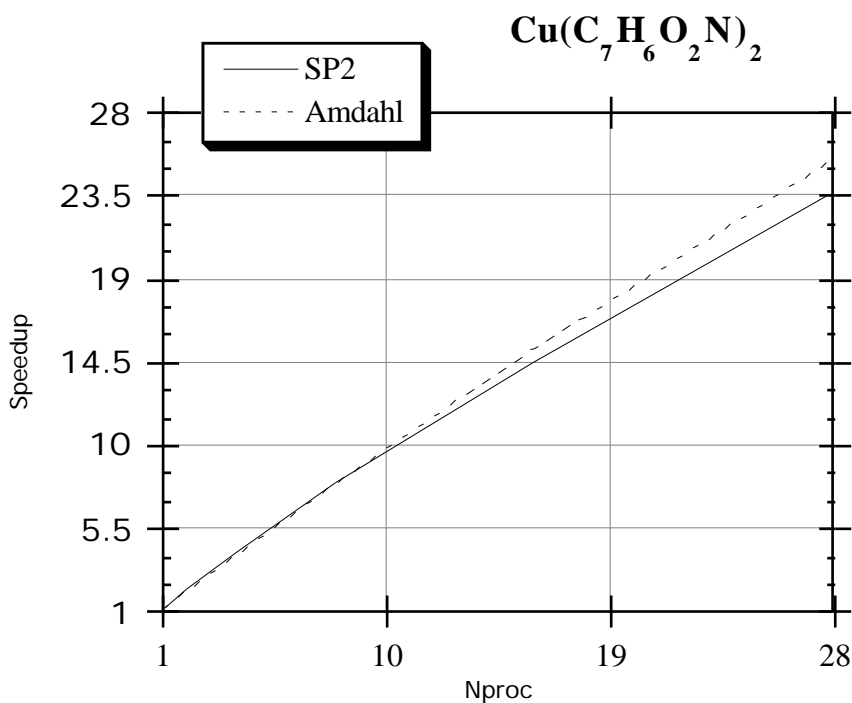
Routine	Speedup	Elapsed time (in sec)
FOCKY	7.8	314
PTCRTN	7.8	773



**Figure 26** Speedup for the routines FOCKY and PTCRTN on the SP1



**Figure 27** The speedups for the cluster and the SP1 and the speedup by Amdahl's law.



**Figure 28** The speedup for the SP2 and the speedup by Amdahl's law.

**Table 8.4** Elapsed time of ADF program for Cu(C<sub>7</sub>H<sub>6</sub>O<sub>2</sub>N)<sub>2</sub>

Parallel platform	Nproc	Speedup	Elapsed time (in sec)
cluster with ethernet	6	5.5	3786
SP1 with FDDI	8	7.3	1404
SP2 with switch	28	23.6	307

In figure 27 the speedup of the whole program on different platforms is shown (see also table 8.4). The parallel part of the program is in this case for a single node run equal to 99.6% of the elapsed time.

The speedup for the workstation cluster is good, 5.5 for the 6-node run. Further we see that the speedup of the IBM SP1 deviates somewhat from Amdahl's law. On the SP2 (see figure 28) this deviation from Amdahl's law is also seen. This is caused by communication.

#### 8.4. Geometry optimisation: Fe<sub>2</sub>(CO)<sub>9</sub>

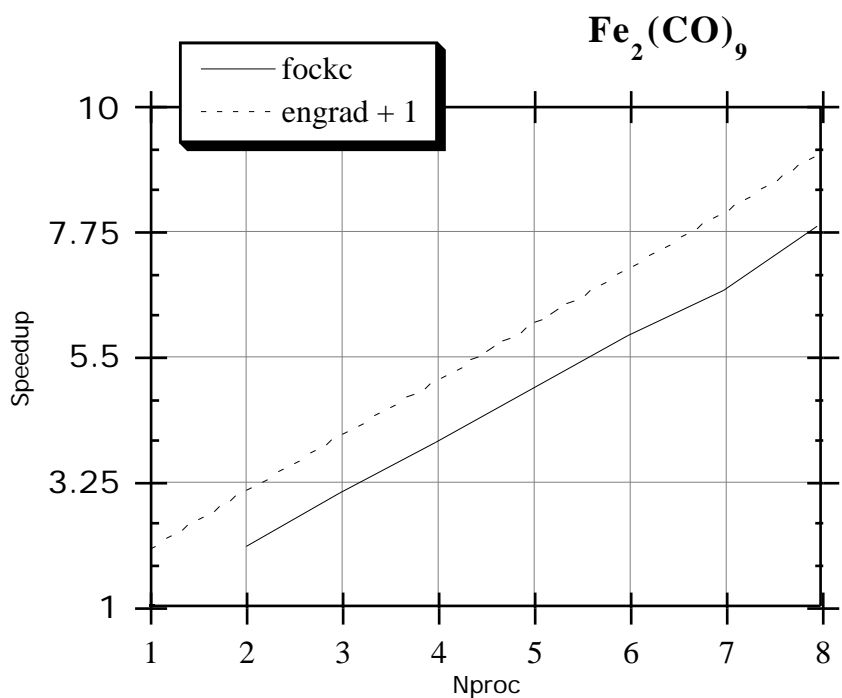
In this section we discuss the speedup for geometry optimisations with gradient corrections. The molecule considered is Fe<sub>2</sub>(CO)<sub>9</sub>. In figure 29 the speedups of the two important routines for the geometry optimisation (FOCKC en ENGRAD) are given. They scale both exactly with the number of nodes (see table 8.5). The load balancing for the numerical integration is perfect and there is no real communication problem.

**Table 8.5** Elapsed time of routines on 8 nodes of the IBM SP1

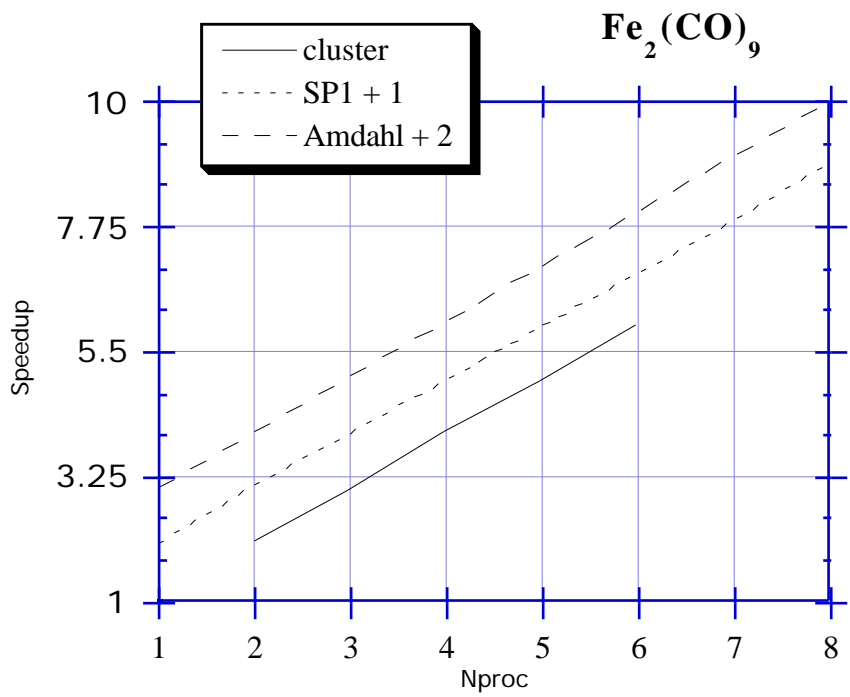
Routine	Speedup	Elapsed time (in sec)
FOCKC	7.9	180
ENGRAD	8.1	1241

**Table 8.6** Elapsed time of ADF program for Fe<sub>2</sub>(CO)<sub>9</sub>

Parallel platform	Nproc	Speedup	Elapsed time (in sec)
cluster with ethernet	6	5.9	29227
SP1 with FDDI	8	7.8	10483
SP2 with switch	28	24.6	2286



**Figure 29** Speedup for the routines FOCKC and ENGRAD on the SP1.

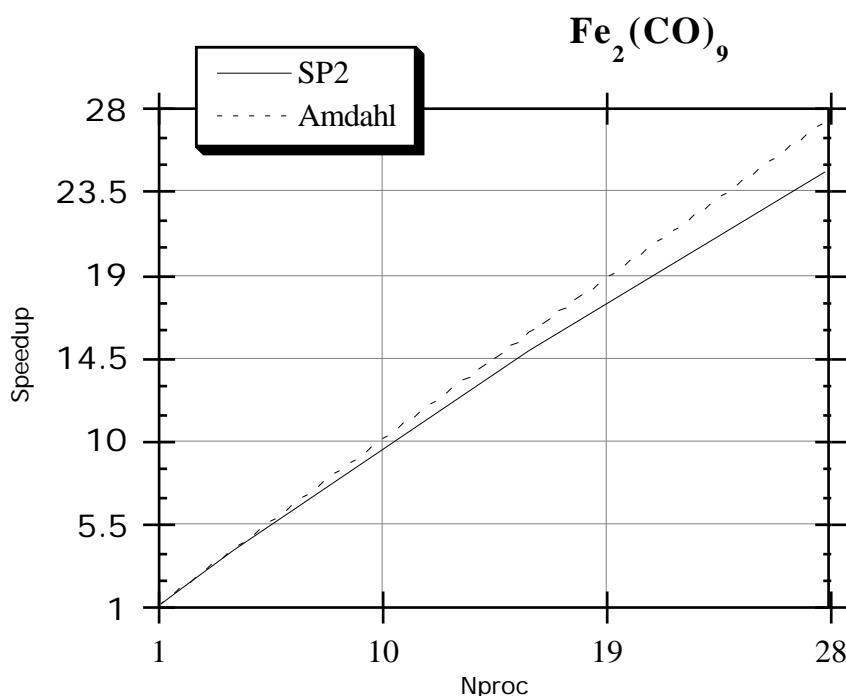


**Figure 30** The speedups for the cluster and the SP1 and the speedup by Amdahl's law.

The speedup of the whole program is shown in figure 30 (see also table 8.6). The parallel part is for a single node run equal to 99.9% of the elapsed time. The measured speedup on the cluster and on the SP1 follow the speedup of Amdahl's law extremely well, so there is no load imbalance and no communication problem at all on both platforms. Thus with a small workstation cluster we are able to reduce an optimisation of one week to one day.

$\text{Fe}_2(\text{CO})_9$  is a rather small molecule compared to the two previous molecules, and also a molecule with a much higher symmetry. So there could be a problem with the load balancing of the routines dealing with the density fit. In fact a very small load imbalance starts to show up for FITINT at the 8 node run. This is caused by the iron-iron pair, which has much more integrals than the other atom pairs. The imbalance for RHOFIH and CONPMT is almost negligible. These three routines only take .7% of the overall elapsed time. So this small load imbalance will not easily cause a decrease of the speedup.

In figure 31 the speedup is shown for the SP2. As in the case of the previous molecule the speedup starts to deviate from Amdahl's law for larger number of nodes. This is also caused by communication. The imbalance in FITINT is almost negligible.



**Figure 31** The speedup for the SP2 and the speedup by Amdahl's law.

## 9. Discussion

From the figures that show the speedups of the individual routines, we see that most of the routines scale with the number of nodes. The three routines (CONPMT, SDIIS and DIAGFM), that do not scale so well with the number of nodes, are also much less time-consuming. The routine DIAGFM will only be parallelised further, when the diagonalisation turns out to be a bottleneck, but we do not expect that it will happen.

We have chosen in our parallelisation strategy to keep the data on file local. The disk space needed by the program scales with the number of nodes. Most of the matrices however have not yet been distributed over the nodes. Therefore the memory size needed will not scale with the number of nodes. From figure 21, where the speedup of RHOFIH is shown, we saw that the distribution of the density matrix decreased the number of page faults dramatically. In the near future we will also distribute the overlap matrix in the same way. Matrices, such as the Fock matrix, that are needed completely on all the nodes, can be distributed, but this will lead to an enormous increase of communication time.

If we look at the speedups of the overall program on the workstation cluster for the three different molecules, we see that a cluster of 6 (or 8) workstations connected by ethernet can be used adequately. Only when more workstations are added to the cluster the communication might become a bottleneck.

On the larger parallel platforms we see that the speedup almost follows Amdahl's law. The small deviation can be caused by load imbalance or communication. It is known that there can be a small load imbalance for the routines that deal with the density fit. However these routines take a very small amount of time, and small deviations in the load balancing will hardly influence the speedup. Therefore the number of nodes used should be conformable to the size of the calculation.

From the speedups of the three molecules shown here, we can conclude that our parallelisation strategy turned out to be a good choice. The parallel version of ADF scales well with the number of processors.

## 10. Acknowledgements

This investigation was supported by the EC through the Europort2 project (ESPRIT-III) and the Holland Research School of Molecular Chemistry.

## 11. References

- 1 P. Hohenberg, W. Kohn, Phys. Rev. **136B**, 864 (1964)
- 2 W. Kohn, L.J. Sham, Phys. Rev. **140A**, 1133 (1965)
- 3 J.C. Slater, Quantum Theory of molecules and Solids, Vol4; McGraw-Hill: New York, 1974
- 4 K.H. Johnson, J. Chem. Phys. **45**, 3085 (1966)
- 5 E.J. Baerends, D.E. Ellis, P. Ros, Chem. Phys. **2**, 41 (1973)
- 6 D.E. Ellis, Int. J. Quantum Chem. 2S, 35 (1968)
- 7 P.M. Boerrigter, G. te Velde, E.J. Baerends, Int. J. Quantum Chem. **33**, 87 (1988)
- 8 G. te Velde, E.J. Baerends, J. Comp. Phys. **99**, 84 (1992)
- 9 H. Sambe, R. Felton, J. Chem. Phys. **62**, 1122 (1975)
- 10 B.I. Dunlap, J.W.D. Connolly, J.R. Sabin, J. Chem. Phys. **71**, 3396 (1979); *ibid* **71**, 4993 (1979)
- 11 A. St Amant, D.R. Salahub, Chem. Phys. Lett. **169**, 387 (1990)
- 12 J. Andzelm, E. Wimmer, J. Chem. Phys. **96**, 1280 (1992)
- 13 S.H. Vosko, L. Wilk, M. Nusair, Can. J. Phys. **58**, 1200 (1980)
- 14 D.M. Ceperley, B.J. Alder, Phys. Rev. Lett. **45**, 566 (1980)
- 15 J.P. Perdew, Phys. Rev. B **33**, 8822 (1986); Erratum Phys. Rev. B **38**, 7406 (1986)
- 16 A.D. Becke, Phys. Rev. A **38**, 3098 (1988)
- 17 A.D. Becke, J. Chem. Phys. **98**, 5648 (1993)
- 18 L. Fan, T. Ziegler, J. Chem. Phys. **94**, 6057 (1991)
- 19 J.P. Perdew, J.A. Chevary, S.H. Vosko, K.A. Jackson, M.R. Pederson, D.J. Singh, C. Fiolhais, Phys. Rev. B **46**, 6671 (1992)
- 20 T. Ziegler, A. Rauk, Theoret. Chim. Acta, **46**, 1 (1977)
- 21 E.J. Baerends, Bond energy evaluation, Internal report, Vrije Universiteit 1988
- 22 B. Delley, A. Freeman, D.E. Ellis, E.J. Baerends, D. Post, Phys. Rev. B **27**, 2132 (1983)
- 23 T. Ziegler, A. Rauk, Inorg. Chem. **18**, 1558 (1979)
- 24 T. Ziegler, A. Rauk, Inorg. Chem. **18**, 1755 (1979)
- 25 J.G. Snijders, E.J. Baerends, Mol. Phys. **36**, 1789 (1978)
- 26 J.G. Snijders, E.J. Baerends, P. Ros, Mol. Phys. **38**, 1909 (1979)
- 27 P. Boerrigter, thesis, Vrije Universiteit, 1988
- 28 T. Ziegler, V. Tschinke, E.J. Baerends, J.G. Snijders, W. Ravenek, J. Phys. Chem.

- 93, 3050 (1989)
- 29 L. Versluis, T. Ziegler, J. Chem. Phys. **88**, 322 (1988)
- 30 L. Fan, T. Ziegler, J. Chem. Phys. **92**, 3645 (1990)
- 31 L. Deng, T. Ziegler, Int. J. Quantum. Chem. **52**, 731 (1994)
- 32 L. Fan, T. Ziegler, J. Chem. Phys. **96**, 9005 (1992)
- 33 L. Fan, T. Ziegler, J. Phys. Chem. **96**, 6937 (1992)
- 34 G. Schreckenbach, T. Ziegler, J. Phys. Chem. **99**, 606 (1995)
- 35 P. Belanzoni, E.J. Baerends, S. van Asselt, P. Langewen, J. Phys. Chem., to be published
- 36 S. van Gisbergen, J.G. Snijders, E.J. Baerends, submitted
- 37 G. te Velde, E. J. Baerends, Phys. Rev. B **44**, 7888 (1991)
- 38 G. te Velde, E. J. Baerends, Chem. Phys. **177**, 399 (1993)
- 39 P. Philipsen, G. te Velde, E.J. Baerends, Chem. Phys. Lett. **226**, 583 (1994)
- 40 J.P. Perdew, Y. Wang, Phys. Rev. B **33**, 8800 (1986)
- 41 Y. Wang, J.P. Perdew, Phys. Rev. B **44**, 13298 (1991)
- 42 C. Lee, W. Yang, R.G. Parr, Phys. Rev. B **37**, 785 (1988)
- 43 E. Engel, S.H. Vosko, Phys. Rev. B **47**, 13164 (1993)
- 44 R. van Leeuwen, E.J. Baerends, Phys. Rev. A **49**, 2421 (1994)
- 45 O. Gritsenko, R. van Leeuwen, E.J. Baerends, J. Chem. Phys. **101**, 8455 (1994)
- 46 R. van Leeuwen, O. Gritsenko, E.J. Baerends, Z. Physik D **33**, 229 (1995)
- 47 O. Gritsenko, R. van Leeuwen, E. van Lenthe, E.J. Baerends, Phys. Rev. A. **51**, 1944 (1995)
- 48 A.D. Becke, J. Chem. Phys. **88**, 2547 (1988)
- 49 K. Kitama, K. Morokuma, Int. J. Quantum. Chem. **10**, 325 (1976)
- 50 E.J. Baerends in: *Cluster Models for Surface and Bulk Phenomena*, Plenum, 1992, p. 189
- 51 F.M. Bickelhaupt, N.N. Nibbering, E.M. van Wezenbeek, E.J. Baerends, J. Phys. Chem. **96**, 4864 (1992)
- 52 J.G. Snijders, Thesis Vrije Universiteit, Amsterdam, 1979
- 53 T. Ziegler, J.G. Snijders, E.J. Baerends, J. Chem. Phys. **74**, 1271 (1981)
- 54 E. van Lenthe, E.J. Baerends, J.G. Snijders. J. Chem. Phys. **99**, 4597 (1993)
- 55 R. van Leeuwen, E. van Lenthe, E.J. Baerends, J.G. Snijders, J. Chem. Phys. **101**, 1272 (1994)
- 56 E. van Lenthe, R. van Leeuwen, E.J. Baerends, J.G. Snijders in "New Challenges in

- Computational Quantum Chemistry" eds. R.Broer, P. Aerts, P.S.Bagus (Groningen 1994) 93
- 57 A.J.Sadlej, J.G.Snijders, Chem. Phys. Lett. **229**, 435 (1994)
- 58 E. van Lenthe, E.J.Baerends, J.G.Snijders, J.Chem.Phys. **101**, 9783 (1994)
- 59 A.J.Sadlej, J.G.Snijders, E. van Lenthe, E.J.Baerends, J. Chem.Phys. **102**, 1758 (1995)
- 60 E. van Lenthe, E.J.Baerends, J.G. Snijders, Chem Phys. Lett. **236**, 235 (1995)
- 61 L. Versluis, T. Ziegler, J. Chem. Phys. **322**, 88 (1988).
- 62 L. Fan, T. Ziegler, J. Chem. Phys. **95**, 7401 (1991).
- 63 P. Pulay, Chem. Phys. Lett. **73**, 393 (1980);  
P. Pulay, J. Comp. Chem. **3**, 556 (1982);  
T.P. Hamilton, P. Pulay, J. Chem. Phys. Lett. **84**, 5728 (1986)
- 64 T. H. Fischer, J. Almlöf, J. Chem. Phys. **96**, 9768 (1992).
- 65 B. A. Murtagh, R. W. H. Sargent, Comp. J. **13**, 185 (1970).
- 66 C. G. Broyden, J.Inst. Maths.Appl. **6**, 76 (1970).
- 67 R. Fletcher, Comp. J. **13**, 317 (1970).
- 68 D. Goldfarb, Math. Comp. **24**, 23 (1970).
- 69 D. F. Shanno, Math. Comp. **24**, 647 (1970).
- 70 W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in Fortran* (University Press, Cambridge, 1992).
- 71 G. Kolata, *Science* **217**, 39 (1982).
- 72 L. Fan, T. Ziegler, J. Am. Chem. Soc. **114**, 10890 (1992).
- 73 J. Baker, J. Comput. Chem. **7**, 385 (1986).
- 74 V. Sunderam, *Concurrency: PracticeExperience* **2**(4), 1990. PVM is available via electronic mail from netlib@ornl.gov.
- 75 E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, 1992
- 76 G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker, *Solving Problems on Concurrent Processors*, volume I, Prentice Hall (1988)
- 77 R.A. van de Geijn, LAPACK Working Note 29, University of Tennessee
- 78 G.M. Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*. In AFIPS Conference Proceedings, 483-485, 1967.