# Draft
# Parallelization of MOPAC

**Gregor von Laszewski**

September 16, 1992

## Contents

# Draft
# Parallelization of MOPAC[1]

**Gregor von Laszewski**
**gregor@npac.syr.edu**

Version 1.3.1

## Abstract

The program MOPAC has been extensively analyzed with the help of test input data to obtain information about program code worth to parallelize. Suggestions for the parallelization are given.

## 1 Introduction

Three main branches of computational chemistry are quite popular:

1. molecular mechanics, where confirmations of macro molecules are studied,
2. ab initio or Gaussian methods, where the properties of small molecules are studied with high accuracy,
3. and semiempirical methods, which are classified to be in-between the two other groups.

The semiempirical methods are the most popular methods. They are in general very slow in comparison to the methods used in molecular mechanics.
MOPAC is a general-purpose, semiempirical molecular orbital program. It enables to study chemical reactions involving molecules, ions, and linear polymers. In contrast to CHARMM, which deals with interactions between molecules and atoms, MOPAC interacts on the electron level. This makes the program more complex and more difficult.
MOPAC includes the four distinct semiempirical methods:

1. MINDO/3,
2. MNDO,
3. AM1,
4. and PM3.

Most of the semiempirical methods are based on a matrix diagonalization. For example, in MNDO a typical calculation accounts 80-85% for the diagonalization.
The four methods are self-consistent field methods (SCF) in which all calculated integrals are approximated by means.
The calculation done uses a restricted basis set of one $s$ orbital and three $p$ orbitals for each atom. Hence, the total number of orbitals is $4N$, where $N$ is the number of atoms. Overlapping integrals are ignored in the secular equation. Therefore, the term

$$|H - E| = 0,$$

is solved, where H is the secular determinant and E is the set of eigenvalues. Because the overlapping integrals are not calculated large systems can be solved in a short time.

## 2 Algorithmic Description Language

Some of the algorithms presented in this paper are described in a high level language based on index sets. An index set is a set which enables access to array elements. The definition of a set is based on mathematical terms. The simplest construct to define a index set is an interval specified by the begin and the end of the range. The mathematical brackets $[,],],[$ are used for this purpose. For example, the index set $[1, 5]$ specifies

[1] This work is sponsored by Digital Equipment and DARPA under contract #DABT63-91-k-0005. The content of the information does not necessary reflect the position or the policy of the Government and no official endorsement should be inferred.

the following set of integers $\{1,2,3,4,5\}$. Note, that there is no order in the set. The operators $=, \cup, \cap$ are defined in the usual mathematical way.
An index set can be modified by a term:

$$[a,b]_i : f(i)$$

The semantic is explained with the help of an example. Let $f(i) = 2i$, than $[1,6]_i : 2i = \{2,4,6\}$. Predefined functions are *even* and *odd* specifying the index sets with only even and odd numbers.
Parallelism on the base on vector operators is expressed with Fortran 90 constructs. Furthermore, the constructs *parallel begin*, *parallel end* and *foreach* are added. The semantic of the constructs is as follows:

```
parallel begin
  statement 1
  ⋮
  statement 2
parallel end
```

The statements in-between parallel begin and parallel are executed in parallel.

```
parallel begin
  statement 1
  ⋮
  foreach i ∈ ]1,n[
    statement(i)
  end foreach
  ⋮
  statement n
parallel end
```

All the statements in the foreach construct can be executed in parallel. In the example shown above the following statements are executed in parallel:

```
statement 1
statement(2)
⋮
statement(n-1)
statement n
```

The brackets should denote that the statements in the loop might be dependent on the incarnation of the loop variable. To express statements which are specifically executed in sequential order the statement *sequential begin* and *sequential end* are used. [7, 8]
Some of the algorithms are presented using message passing. Therefore the SEND and RECEIVE statements are used. The command

SEND message TO processor$_i$

sends the message to the processor with the number $i$. On the other hand

RECEIVE message FROM processor$_i$

receives a message from processor with the number $i$. Is the processor identification unimportant for the algorithmic specification, only

RECEIVE message

should be used. A broadcast or a message to a subset of processors is possible via the command

SEND message TO all processors

and, for example,

SEND message TO all processors with even processor ID.

## 3 The Program MOPAC

The version of MOPAC used for the study is version 6.0 (the newest version). The program MOPAC is quite large with approximately 59000 lines (Table 1).

It is developed by a large number of experts in computational chemistry. Therefore, it is difficult for a non chemist to understand the computational part of the program.

| | Number |
|---|---|
| Lines (total) | 59006 |
| Statements | 40846 |
| Comment lines | 13386 |
| Number of procedures | 324 |
| Number of functions | 39 |

Table 1: Analysis of the type of lines in the program MOPAC

MOPAC is a single program, written in Fortran77. Analysis of the program code shows that the code is almost pure Fortran77. The documentation for the user is sufficient [1]. The documentation available for the numerical calculations done in the program varies from routine to routine but is supported by other publications for the chemical expert [10, 2]. Therefore, one needs to understand the principal of the chemical calculations in order to make use of

- the comments given within the code
- and the publications related to the program.

An advantage of MOPAC is its availability in public domain. Updates are done on a regular basis every year. MOPAC is available on a number of platforms including Sun Sparc and Cray-2. This shows that the program is

- still supported
- and quite popular.

Although the manual states out that the code has been optimized for vector-computers, the optimization is mentioned in only a few routines of the source code. These, routines are matrix and vector operations. The program analysis stated out that they are not often used. Hence, these optimizations are not useful to support a parallelization on a MIMD machine.

### 3.1 Geometry Optimization

MOPAC uses an internal representation different from Cartesian coordinates to do geometry minimizations. Nevertheless, Cartesian coordinates are supported as option for input and output. The method used to do the optimization is based on the derivatives of the energy with respect to coordinates. The geometry is changed so as to lower the heat of the formation. In case that no further changes can lower the heat of formation significantly, the optimization is terminated. Now the geometry corresponds to a stationary point on the potential surface. Generally this will be one of many possible conformers or isomers.[2]
The termination criteria is

1. the predicted change in the heat of formation
2. the current gradient norm

### 3.2 Precision and Accuracy

The Program is only as accurate as its underlying model which is given by the authors of the program. The precision of the calculation can be determined by a number of parameters used for the specific model.[3]

### 3.3 Control Within Mopac

Each subroutine is made independent, as far as possible, even at the expense of extra code or calculation. Important variables are generally modified only in one routine. When a subroutine is called, it assumes that all data required for its operation is available in either

[2]This points out the problem of evaluation of correctness of the program calculation. A chemist might be able to evaluate if the result is correct.

[3]However,in order to specify the parameters in a useful way the interaction with a chemist is necessary.

common blocks or arguments. All data is *exclusively* owned by exactly one subroutine. Only this subroutine gets permission and ability to change the data. This implies that parallelization between different subroutines should be avoided. Thus, it is sufficient to concentrate on kernel subroutines used in MOPAC.
A general subroutine (for example ITER) handles three kinds of data:

1. data which the subroutine is going to work on, for example the one and two electron matrices.
2. data necessary to manipulate the first set of data, such as the number of atomic orbitals.
3. the calculated quantities, here the electronic energy, and the density and Fock matrices.

# 4 Experiments

Two experiments with different input data are analyzed in this section in order to find computational insensitive parts for parallelization.
The different experiments showed that roughly half of the program code is used to obtain the result. Only a few of the used routines are computationally very intense. These routines will be studied in a later section in more detail.
The first experiment is an optimization of a Formaldehyde molecule with 4 atoms and the second one of a Crystal with 48 atoms (11-cis retinal). For the Formaldehyde calculation MNDO has been chosen and for the crystal AM1 is used.
The calculation using the Formaldehyde could verify the correctness of the program. Further tests are necessary to verify the functionality of the code running on the Sun Sparc. [4]
The logic of the program package MOPAC is displayed in Figure 1. The program consists of two main sequences, geometric and electronic, which join only at the subroutine COMPFG. The keywords shown in the flowchart specify logical units or subroutines defined in the source code. The calculation starts at the main routine. The ordinate represents the time axis. Movements on the edges of the flow chart are only allowed in horizontal or down direction. A short description of most of the routines used in the program can be found in [1].
An interesting problem would be the calculation on flexible molecules which might change their appearance and geometrical structure drastically. This means beside small changes in the geometrical appearance also catastrophic events can take place.

[4] Due to the fact of a warning during compilation which could not be avoided further checks are necessary.

## 4.1 Formaldehyde

The first test case is a MNDO calculation on formaldehyde (Figure 2) to calculate the ground state. The input data for MOPAC is shown in Figure 3.

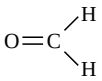


Figure 2: The stick diagram of the Formaldehyde molecule

FORCE T=2500 NOINTER VECTORS PRECISE
formaldehyde test input for mopac
ground state with many options

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| O | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0 | 0 | 0 |
| C | 1.30 | 1 | 0.00 | 0 | 0.00 | 0 | 1 | 0 | 0 |
| H | 1.10 | 1 | 120.00 | 1 | 0.00 | 0 | 2 | 1 | 0 |
| H | 1.10 | 1 | 120.00 | 1 | 180.00 | 1 | 2 | 1 | 3 |

Figure 3: The input data for the Formaldehyde to the MOPAC program

To show the time spend in the calculation, a simple geometry optimization of a Formaldehyde molecule has been done. The values in the flowchart (Figure 1) represent the the percentage of the total time of the program accounted for by this function and its descendants. The Table 2 shows some of the most time consuming routines of the program. *Internal calls* are

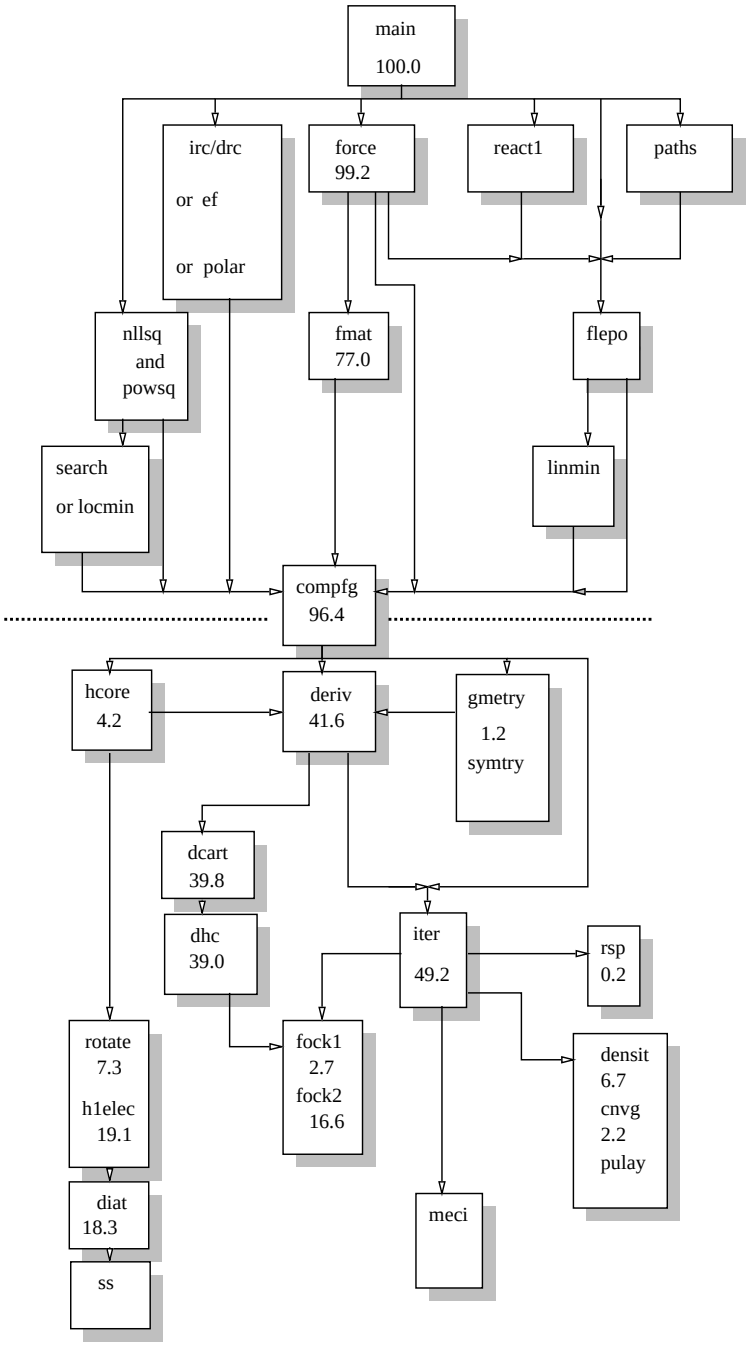


Figure 1: Flowchart of the electronic sequence of the program MOPAC using Formaldehyde as input data

calls of routines to open files, to do the book keeping of the analysis program and also include functions like multiplication and division. The other routines are routines within the program.
The columns in the Table 2 are specified as follows:

**time** is the percentage of the total running time of the program used by this function.

**scaled time** is the percentage of the total running time of the program used by this function without taking the time used to do the book keeping with the help of the function *mcount* This is a more realistic view of the total running time.

**cumulative seconds** is the running sum of the number of seconds accounted for by this function and those listed above it.

**self seconds** is the number of seconds accounted for by this function alone.

**calls** is the number of times this function was invoked.

**self ms/call** is the average number of milliseconds spent in this function per call.

**total ms/call** is the average number of milliseconds spent in this function and its descendants per call.

**name** is the name of the function.

A short description of these routines is given in the Tables 5 and 6 given in the appendix [1]. For the calculation with the Formaldehyde a lot of the time is spend in the diagonalization and the calculation of the Fock matrix.

### 4.2 Crystal (11-cis retinal)

The calculation done on the crystal involves 48 atoms and uses the AM1 calculation. The result is somewhat different from the Formaldehyde calculation. Here no geometry calculation is done at all. For this calculation most of the time is spend in the diagonalization and the calculation of the density matrix.

## 5 SCF Calculation in MOPAC

The core of the calculation consists of repeated application of a diagonalization routine. In order to speed up the calculation one starts first with a matrix which is close to its diagonalform and performs a fast diagonalization routine. The last step of the iteration uses an exact diagonalization routine.
The names for the routines are *diag* for the fast diagonalization routine and *hqrii* for the exact one.
For a complexity analysis the abbreviations shown in Figure 5 are used.

| | |
|---|---|
| $N$ | number of atoms $N = n_h + n_l$ |
| $n_h$ | number of heavy atoms such as C |
| $n_l$ | number of light atoms such as H |
| $n$ | number of orbitals $n = n_o + n_v = 4N$ |
| $n_o$ | number of occupied orbitals |
| $n_v$ | number of virtual orbital |

Figure 5: The main parameters which determine the complexity of the algorithm

### 5.1 The Pseudo Diagonalization

The diagonalization routine is used on a good starting approximation and/or a few steps of a conventional SCF iteration, where the Fock matrix has been brought to approximately diagonal form in orbital basis [10].
The procedure DIAG is a *fast* pseudo-diagonalization procedure, in that the vectors that are generated by it are more nearly able to block-diagonalise the fock matrix over molecular orbitals than the starting vectors. it must be considered pseudo for several reasons [9]:

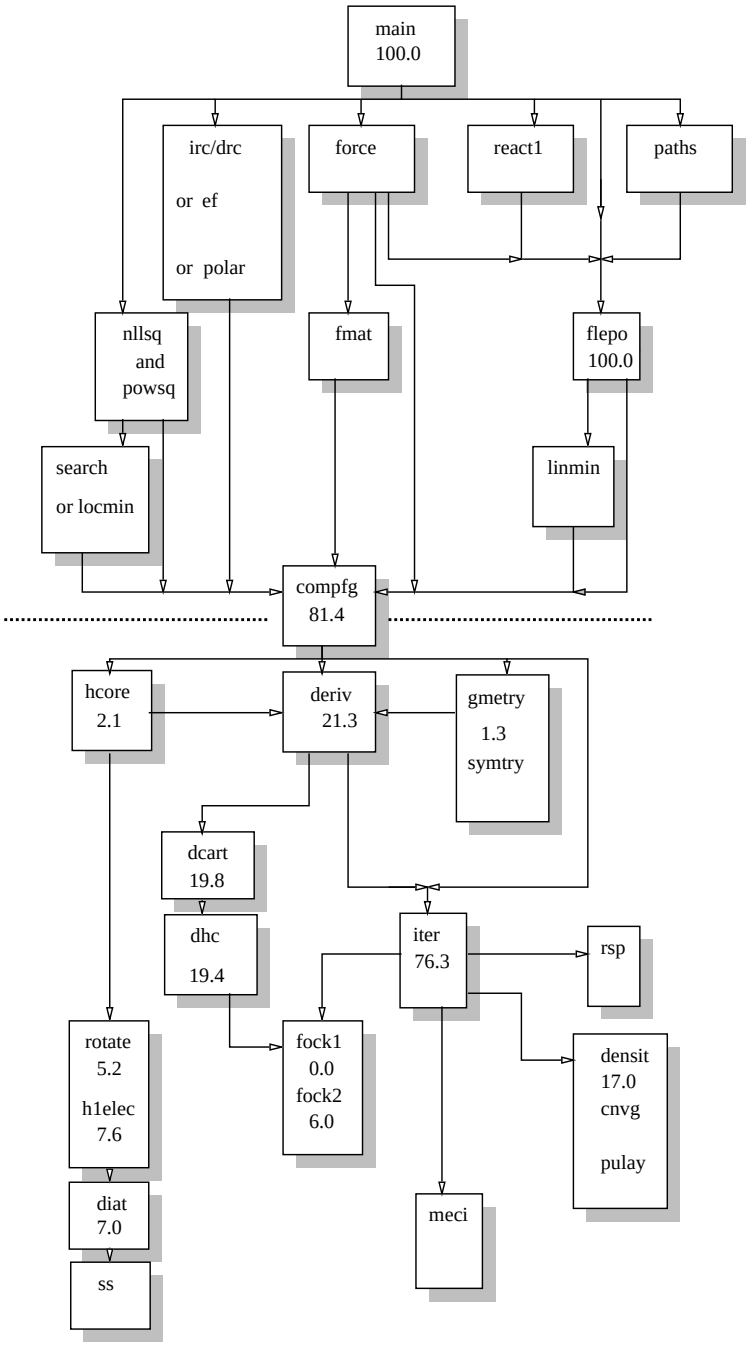


Figure 4: Flowchart of the geometric and electronic sequence of MOPAC using the crystal data as input (11-cis retinal)

Internal calls and program analysis routines

| scaled time | time | cumulative seconds | self seconds | calls | self ms/call | total ms/call | name |
|---|---|---|---|---|---|---|---|
| 0.0 | 12.6 | 0.49 | 0.49 | | | | mcount |
| 7.6 | 6.7 | 1.43 | 0.26 | | | | odd |
| 8.8 | 7.7 | 1.17 | 0.30 | 84 | 3.57 | 3.57 | open |
| 2.4 | 2.1 | 2.77 | 0.08 | 132977 | 0.00 | 0.00 | .mul |
| 1.4 | 1.0 | 3.32 | 0.04 | | | | mul4bit |

| scaled time | time | cumulative seconds | self seconds | calls | self ms/call | total ms/call | name |
|---|---|---|---|---|---|---|---|
| 11.2 | 9.8 | 0.87 | 0.38 | 62 | 6.13 | 7.07 | diag |
| 7.6 | 6.7 | 1.69 | 0.26 | 335 | 0.78 | 1.30 | fock2 |
| 5.8 | 5.1 | 1.89 | 0.20 | 7 | 28.57 | 30.80 | hqrii |
| 5.2 | 4.6 | 2.07 | 0.18 | 300 | 0.60 | 1.58 | diat |
| 5.2 | 4.6 | 2.25 | 0.18 | 300 | 0.60 | 1.03 | rotate |
| 3.5 | 3.1 | 2.37 | 0.12 | 541 | 0.22 | 0.22 | iindx |
| 3.2 | 2.8 | 2.48 | 0.11 | 300 | 0.37 | 0.42 | repp |
| 3.2 | 2.8 | 2.59 | 0.11 | 67 | 1.64 | 1.91 | densit |
| 2.9 | 2.6 | 2.69 | 0.10 | 125 | 0.80 | 0.80 | jab |
| 2.4 | 2.1 | 2.85 | 0.08 | 8994 | 0.01 | 0.01 | powdi |
| 2.4 | 2.1 | 2.93 | 0.08 | 252 | 0.32 | 4.45 | dhc |
| 2.1 | 1.8 | 3.00 | 0.07 | 734 | 0.10 | 0.18 | bintgs |
| 2.1 | 1.8 | 3.07 | 0.07 | 300 | 0.23 | 0.24 | coe |
| 2.1 | 1.8 | 3.14 | 0.07 | 125 | 0.56 | 0.56 | kab |
| 1.5 | 1.3 | 3.19 | 0.05 | 343 | 0.15 | 0.15 | helect |
| 1.5 | 1.3 | 3.24 | 0.05 | 67 | 0.75 | 0.75 | cnvg |
| 1.1 | 1.0 | 3.28 | 0.04 | 300 | 0.13 | 0.74 | diat2 |

Table 2: Running times using the formaldehyde data as input

- it does not generate a complete set of eigenvectors - the secular determinant is not diagonalised, only the occupied-virtual intersection. That means only the eigenvectors for the occupied orbitals are generated.

- many small elements in the secular determinant are ignored as being too small compared with the largest element.

- when elements are eliminated by rotation, the rest of the secular determinant is assumed not to change, i.e. elements created are ignored.

- the rotation required to eliminate those elements considered significant is approximated to using the eigenvalues of the exact diagonalization throughout the rest of the iterative procedure.

The procedure has the following input parameters:

| Parameter | Description |
|---|---|
| $A_{packed}$ | contains the lower half triangle of the matrix to be diagonalised in a packed format. |
| vector | contains the old eigenvectors on input, the new vectors on exiting. |
| $n_o$ | number of occupied molecular orbitals. |
| eig | eigenvalues from an exact diagonalization |
| $n$ | number of atomic orbitals in the basis set |

Internal calls and program analysis

| scaled time | time | cumulative seconds | self seconds | calls | self ms/call | total ms/call | name |
|---|---|---|---|---|---|---|---|
| 0.0 | 26.2 | 1889.94 | 898.42 | | | | mcount |
| 11.5 | 8.5 | 2180.05 | 290.11 | 37953170 | 4 0.0 | 0 0.00 | .mul |
| 9.1 | 6.7 | 2692.50 | 229.20 | | | | mul8bit |
| 3.1 | 2.3 | 2771.93 | 79.43 | | | | odd |
| 1.2 | 0.9 | 3123.07 | 31.88 | | | | mul4bit |
| 0.7 | 0.5 | 3303.96 | 15.59 | | | | zerodivide |

| scaled time | time | cumulative seconds | self seconds | calls | self ms/call | total ms/call | name |
|---|---|---|---|---|---|---|---|
| 39.3 | 29.0 | 991.51 | 991.51 | 147 | 6744.98 | 8128.06 | diag |
| 11.2 | 8.3 | 2463.30 | 283.25 | 150 | 1888.34 | 2420.42 | densit |
| 2.8 | 2.1 | 2842.69 | 70.76 | 124656 | 0.57 | 0.90 | rotate |
| 2.7 | 2.0 | 2911.93 | 69.24 | 106022 | 0.65 | 1.20 | fock2 |
| 2.2 | 1.6 | 2967.52 | 55.59 | 124656 | 0.45 | 1.20 | diat |
| 1.9 | 1.4 | 3017.05 | 49.53 | 4 | 12382.53 | 13671.00 | hqrii |
| 1.5 | 1.1 | 3054.92 | 37.87 | 124656 | 0.30 | 0.31 | repp |
| 1.5 | 1.1 | 3091.19 | 36.28 | 105840 | 0.34 | 3.90 | dhc |
| 1.2 | 0.9 | 3154.08 | 31.01 | 57120 | 0.54 | 0.55 | jab |
| 1.2 | 0.9 | 3183.29 | 29.21 | 124656 | 0.23 | 0.24 | coe |
| 0.9 | 0.7 | 3208.76 | 25.47 | 57120 | 0.45 | 0.45 | kab |
| 0.9 | 0.7 | 3233.34 | 24.58 | 4270 | 5.76 | 6.38 | gmetry |
| 0.8 | 0.6 | 3252.85 | 19.51 | 113353 | 0.17 | 0.57 | diat2 |
| 0.7 | 0.5 | 3270.89 | 18.05 | 234972 | 0.08 | 0.11 | bintgs |
| 0.7 | 0.5 | 3288.37 | 17.48 | 106038 | 0.16 | 0.16 | helect |

Table 3: Running times using the crystal data as input (11-cis retinal)

The first part of the diagonalization routine constructs the secular determinant over molecular orbitals which connects occupied and virtual sets. The sequential algorithm uses a triangular matrix as input parameter in order to save memory. For the parallelization of this sequential algorithm it is better to keep the whole matrix even though more memory is used. Substituting the triangular matrix by its complete counterpart enables one to rewrite the sequential algorithm in the way shown in Figure 6.

In 6 the vector $w$ is a temporary variable, $fmo$ specifies the density matrix in triangular form, and vector(:,i) is the $i^{th}$ column vector of the matrix *vector* of length $n$. The complexity of calculating the occupied-virtual block of the Fock matrix as shown above is

$$O(n_v n^2 + n_o n_v n)$$

The parallelization can be done easily on $p$ processors such that the complexity of this part of the diagonalization routine is

$$O(\frac{n_v n^2 + n_o n_v n}{p})$$

The parallel algorithm is scalable with the number of processors assuming that

$$p << n$$

and therefore no processor is idle.

```
round = 0;
foreach j ∈ {occupied orbitals} do parallel
  sequential begin
    foreach i ∈ {all orbitals} do parallel
      w(i) = A(:,i) * vector(:,j)
    end foreach
    foreach k ∈ {virtual orbitals} do parallel
      fmo(n_o *round + 1)
         = w * vector(:,n_o)
    end foreach
    round = round + n_o
  end sequential
end foreach
```

Figure 6: Construction of the secular determinant

The second part of the diagonalization is basically a Jacobi transformation. The Jacobi Method is an iterative method for determining eigenvectors [6, 12]. The idea behind the Jacobi method is to do simple transformations on the matrix in order to make the matrix more diagonal. Since one can perform the transformations independently this method is well suited for parallelization.
The basic outline of the sequential algorithm is given in Figure 7.

```
while NOT terminate
   a_ij = the largest matrix element
   transform the matrix on the columns
      and rows of a_ij
end
```

Figure 7: sequential Jacobi method

Since the transformation is done in $O(n)$ steps the complexity is determined by finding the maximum element which is clearly an $O(n^2)$ problem. $N_s$ Jacobi updates are referred as a sweep, where $N_s$ is of the size of the triagonal matrix $n(n-1)/2$. Heuristically the Jacobi method needs $O(\log n)$ sweeps. Therefore, the complexity is

$$n^3(n-1)/2\log n$$

Since the search of the maximal element takes such a long time the algorithm can be modified by using a cyclic-by-row strategy. Instead of looking for the maximal element the matrix is modified a fixed order. With this modification one needs

$$n^2(n-1)/2\log n$$

steps. By parallelizing this method one can distribute several rotation steps on different processors so that the complexity is

$$n^2(n-1)/2\log n/p$$

This is still larger than Householder transformations. In contrast to the Jacobi method, which appears for sequential computers very inefficient in comparison to the Householder method, the transformations in the SCF calculation are not iterated. Furthermore, only the elements of the matrix are annihilated if they are larger than a threshold value of about 0.04. The loop is executed over all pairs of virtual to occupied orbitals. The algorithm can than be specified as shown in Figure 8.

```
foreach i ∈ {occupied orbitals}
  foreach j ∈ {virtual orbitals}
    if (a_ij > threshold) then
      transform the matrix on the columns
          and rows of a_ij
    end if
  end foreach
end foreach
```

Figure 8: Parallel Conditional Jacobi method in the SCF calculation

The number of multiplications used in the sequential

transformation step is

$$4n_o n_v n$$

since $4n$ multiplications are used in the transformation step. Under the assumption that $n_o = n_v = n/2$ the total number of multiplications used in the diagonalization routine is less than $2n^3$. In contrast the Householder method is about four times more expensive. Nevertheless, one has to take into account additional time for matrix modifications which makes it up to 2-3 times faster than the Householder method for one iteration.

Distributing the load equally over the processor is here more complicated since the condition in the if statement is determined at runtime. One way to avoid this problem is to argue heuristically and distribute the pairs of (i,j) randomly over the processors this should distribute the load equally over the processors in case of large matrices. In case of smaller matrices it is not worth to think about a more complex load distributor since the time spend for distributing the load would unnecessarily decrease the program speed. Only if the mapping scheme plus the execution time used for the calculation is smaller than for the simple random distribution a decrease in the running time is possible.

## 5.2 The Standard Eigenvalue Diagonalization

The routine *hqrii* is able to solve standard eigenvalue problems with the help of the *Householder-QR-Inverse Iteration* method. The routine is divided into three mayor steps reflecting the name of the routine.

- (H) the dense matrix A is converted into the tridiagonal form T with the help of the Householder algorithm.
- (QR) all eigenvalues of T are determined by the QR algorithm.
- (II) Some of the eigenvectors are found with the help of the inverse iteration algorithm where Gaussian elimination with partial pivoting is used. The eigenvectors of A are obtained as the product of H and the eigenvalues, where H is the Householder transformation matrix.

The transformation of the matrix into triangular form is the major difference to the Jacobi-method. Now this modified matrix can be used to to find eigenvalues efficiently with the QR-method. Only the eigenvectors for the occupied orbitals are calculated.

## 5.3 Parallelization of the Sequential Algorithm

Before describing the single steps of the calculation it is useful to compare the overall complexity of the single steps as shown in Table 4. From this table it is clear that the Householder transformation to a triagonal matrix is the most expensive part of the sequential algorithm. The following sections outline the sequential algorithm and the parallelization of it.

| Step | Operations |
|---|---|
| Householder transformation | $\frac{2}{3}n^3 + \frac{3}{2}n^2$ |
| QR-iteration | $9n^2$ |
| Calculating $n_o$ eigenvectors | $10n_o n$ |
| Transforming $n_o$ eigenvectors | $n_o n(n-1)$ |
| Total steps | $\frac{2}{3}n^3 + \frac{21}{2}n^2 + n_o n^2 + 9n_o n$ |

Table 4: Computational steps of the sequential Householder iteration

.

### 5.3.1 Householder Reduction

The Householder reduction uses $n-2$ transformations of the following form:

$$A_{k+1} = H_k^T A_k H_k$$

where $H_k$ is denotes the k$^t h$ transformation. Because

the transformation are real and symmetric, $H_k^T = H_k$. $H_k$ is chosen as

$$P_k = \begin{bmatrix} I' & 0 \\ 0 & (I - ww^T)_k \end{bmatrix}$$

where $ww^T = 1$. One way to obtain the matrix $A_{k+1}$ is to do the following step:

$$A_{k+1} = A_k - vz^T - zv^T$$

where

$$v = 2rw, z = u - w^T uw, u = 1/rA_k w$$

and the choice of w, r as given in the detailed algorithm

```
    do k = 1, n − 2
      k1 ← k + 1
(1)   calculating the norm
      T ← sqrt (Σ_{i=k1}^{n} a_{ki}^2)
(2)   calculating u and v
      if a_{k,k1} < 0 then T ← −T
      a_{k,k1} ← a_{k,k1} + T
      R ← a_{k,k1} * T
      v ← a(k, k1 : n)
      a_{k,k1} ← −T
(3)   do i = k1, n
        u_i ← Σ_{j=i}^{n} a_ij * v_j
      end do
      do i = k + 2, n
        u_i ← u_i + Σ_{j=k1}^{n} a_ji * v_j
      end do
      u ← u / R
(4)   C ← Σ_{i=k1}^{n} v_i * u_i
(5)   calculating z
      z ← u − (C/2R) * v
(6)   calculating A_{k+1}
      do i = k1, n
        do j = i, n
          a_ij ← a_ij − v_i z_j − z_i v_j
        end do
      end do
```

Figure 9: Sequential Householder algorithm

Clearly this algorithm is of complexity $O(n^3)$.
Since the algorithm is inherently sequential because each transformation step has to be completed before the next one is starting, it is possible to incooperate parallelism in the $n-2$ transformation steps. One approach is decomposing the matrix in a square as shown in [5]. Figure 10 shows an example of a square decomposition of a matrix with $D \times D$ elements onto 49 processors. In this decomposition each processor holds a submatrix of size $s \times s$.

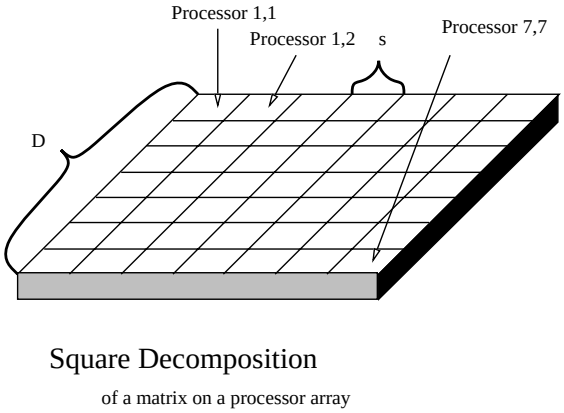


Figure 10: Square Decomposition of a matrix with $D \times D$ elements onto 49 processors. Each processor holds a submatrix of size $s \times s$.

Therefore, the element $a_{ij}$ of the matrix is hold by the processor $(w(i), w(j))$, where

$$w(i) = mod(i-1, D) + 1$$

Since each processor stores only a submatrix of $A$, denoted by $G$, the matrix element $a_{ij}$ is stored in processor $(w(i), w(j))$ in the submatrix element $g_{p(i),p(j)}$ with

$$p(i) = (i-1)divD + 1.$$

The operator $div$ specifies the integer division without rest and $mod$ gives the remainder of the division. The functions

$$W(i) = (i-1)D + r$$

$$e(i) = (i-1)D + c$$

mappes a submatrix element $g_{ij}$ back into A, namely $a_{W(i),e(i)}$ dependent on the row and column of the processor. With this notational help a message passing algorithm can be formulated as shown in Figure12.
The basic idea of the algorithm above is to distribute the calculation of the vectors v, u, z and the scalars C, T and the update of the matrix in each step over the processing elements. Unfortunately, the degree of parallelism is limited since many processors are idle in different steps. of the algorithm. For example, while calculating the norm only the processing elements containing the $k^{th}$ row of the matrix a are involved in the computation.
In a detailed analysis of the algorithm one can see that the efficiency of the algorithm is by omitting terms with low magnitude

$$e = \frac{2/3}{1 + 3/s + \frac{Q_2}{2sQ} + \mu(\frac{5D^2}{2s} - \frac{5D}{2s})}$$

where Q specifies the time used for one multiplication and one addition, and $Q_2$ specifies the time for one multiplication, D additions and 3 subtractions, and where $\mu = \frac{\text{communication time for one datum}}{Q}$.
The number of processor used for this algorithm should be greater or equal to 16 and is approximately 66% [5]. If one incooperates more communication in the last step of the algorithm as shown in [5, 3], the efficiency is 83% if the time for sending the data is small and takes approximately 1/10 of the time of an addition.

#### 5.3.2 QR-iteration

The QR iteration can be parallelized as shown in [4].

[5] assuming that the times for multiplication and addition are normed

#### 5.3.3 Eigenvector Calculation

The calculation of the Eigenvectors can be done in $O(n^2)$. Once the calculation of the eigenvalues is completed, the required eigenvectors are calculated using simple inverse iteration with shift. First the eigenvectors are calculated with the following computational step, where $i \in \{1, 2, ..., n_o\}$:

$$(H_{n-1} - \lambda I)x_i^{(k+1)} = x_i^{(k)} \quad (1)$$

Now the eigenvectors have to be transformed into

$$\phi_i = H_1 ... H_n - 2x_i^{(3)} \quad (2)$$

Since the Householder transformation is already known the Figure 13 shows the parallelization of both steps. Hence, the maximal possible parallelism is determined by $n_o$. In each iteration $10n + n(n-1)$ steps are executed.
Another way to parallelize this problem might be better suited. Instead of calculating the eigenvectors in parallel it might be better to parallelize the Gaussian elimination process. This is for example shown in [11]. Than the eigenvectors are calculated one after another.

## 6 Conclusion

The program MOPAC has been analyzed sequentially in order to find parts in the code which are worth to parallelize. For typical geometry optimization calculation the most time consuming part is the diagonalization of the density matrix, even though a fast diagonalization routine is already used. The fast diagonalization routine applied to triagonal matrices can be easily parallelized.
In order to have a precise diagonalization procedure an exact diagonalization routine is applied. This routine is based on householder transformations and as shown in [4, 5] a parallelization routine can be formulated.
Since this is an ongoing project it is to be expected

that more parts are added to the report.

## Acknowledgment

I would like to thank Professor Birge from Chemistry Department at Syracuse University for his comments and the the sample input data. My special thanks goes to Professor Fox, Professor Ranka and Dr. Haupt at NPAC who helped me in understanding more about the problem.

## 7 Appendix

Figure 15 shows the input data for the Crystal calculation (11-cis retinal).

## References

[1] Michael B.Coolidge and James J. P. Stewart. *MOPAC Manual.* Frank Seiler Research Laboratory,, United Air Force Academy, CO 80840, dec 3100 edition edition, December 1990.

[2] Y. Beppu. HQRII: A Fast Diagonalization Subroutine. *Computers and Chemistry*, Vol. 6(No. 2):pp. 87–91, 1982.

[3] H. Y. Chang, S. Utku, M. Salama, and D. Rapp. A parallel Housholder Tridiagonalization Strategem using scattered square decomposition. *Parallel Computing*, 6:pp. 297–311, June 1988.

[4] G. C. Fox. Eigenvalues of Symmetric Triagonal Matricies. Technical Report $C^3P$ 95, Cal Tech, December 1984.

[5] G. C. Fox. Square Matrix Decompositions: Symmetric, Local, Scatterd. Technical Report $C^3P$ 97, Cal Tech, August 1984.

[6] G. H. Golub and C. F. Van Loan. *Matrix Computations.* John Hopkins University Press, 1989.

[7] G. C. Fox S. Hiranadani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu. Fortran D Specification. Technical Report Rice Comp TR90-141, Rice University, Apr 1991.

[8] C.A. Hoare. *OCCAM 2 Reference Manual.* Seiries in Computer Science. Prentice Hall, New York, 1988.

[9] J. Stewart. MOPAC 6.0 Source Code, SUN. anonymous ftp from ouchem.chem.oakland.edu: /pub/mopac, 1992.

[10] J.J.P. Stewart, P. Cászár, and P. Pulay. Fast Semiempirical Calculations. *Journal of Computational Chemistry*, Vol. 3(No. 2):pp. 227–228, 1982.

[11] G. von Lasewski, M. Parashar, A. G. Mohamed, and G. C. Fox. High Performance Scalable Matrix Algebra Algorithms for Distributed Memory Architectures. In *Proc. of Supercomputing.* to be published, October 1992.

[12] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Oxford Science Publications, 1988. c1965.

```
do k = 0, n − 3
  k ← k + 1
  k_1 ← k + 1
  if processor contains k^th row then
    T^c = Σ j = 1, e(j) > k^s (g_{p(k),j})^2
    SEND T^c TO processor containing a_{k,k1}
  endif
  if I am processor containing a_{k,k1} THEN
    do c = 1, D
      RECEIVE T^c
    T = √( Σ_{c=1}^{D} T^c )
  end if
  if processor contains k^th row then
    v_j^1 = g_{p(k),j}        j = 1...s, e(i) > k
  end if
  if I am processor containing a_{k,k1} THEN
    v_{p(k_1)}^1 = v_{p(k_1)}^1 + sgn (v_{p(k_1)}^1) |T|
    R = v_{p(k_1)}^1 T
    g_{p(k),p(k_1)} = −T
  end if
  if processor contains k^th row then
    SEND v_i^1 TO processors in w(p(i))^t h column
        ∀_i p(i) > k
  endif
  if processor contains w(p(i))^t h column then
    RECEIVE v_i^1
  endif
  if P is on diagonal then
    v_i^2 ← v_i^1          i = 1, ..., s p(i) > k
    SEND v_i^2 TO processors in row w(W(i))
        ∀ W(i) > k
  endif
  if P contains row w(W(i)) then
    RECEIVE v_i^2 s
  endif
  if processor contains a_{k,k_1} then
    SEND R to diagonal processors
  endif
  if I am diagonal processor then
    Receive R
  end if
  u_i^1 = Σ_{j=1,p(j)>k}^{s} g_{ij} v_j^1     i = 1, ..., s p(i) > k
  SEND u^1 TO diagonal processor
  if I am diagonal processor then
    do c = 1, D
      RECEIVE u^1
    u_i^1 = 1/R Σ c = 1^D (u_i^1)^c (?)  i = 1, .., s, W(i) > k
    C^r = Σ_{i=1,p(i)>k}^{s} v_i^1 u_i^1
    SEND C^r TO processor containing a_{kk}
  end if
```

Figure 11: Parallel hqrii algorithm for a message passing platform

```
  if processor contains a_{kk} then
    do r = 1, D
      RECEIVE C^r
    C ← Σ r = 1^D C^r
    Y ← C/2R
    SEND Y TO diagonal processors
  end if
  if I am a diagonal processor then
    RECEIVE Y
    z_i^2 ← z_i^1 ← u_i^1 − Y v_i^1   i = 1, ..., s, W(i) > k
    SEND z_i^1 TO processors in w(e(i))^t h column
        ∀_i e(i) > k
    SEND z_i^2 TO processors in w(W(i))^t h column
        ∀_i W(i) > k
  end if
  if I am in w(e(i))^t h column or w(W(i))^t h column
    RECEIVE z_i^1 or z_i^2
  end if
  g_{ij} ← g_{ij} − v_i^2 z_j^1 − z_i^2 v_j^1
      i = 1, ..., s   e(j) > k
      j = 1, ..., s   W(i) > k
end do
```

Figure 12: Parallel hqrii algorithm for a message passing platform (continued)

```
do i = 1, n_o in parallel
  (H_{n−1} − λ I) x_i^{(k+1)} = x_i^{(k)}
  φ_i = H_1 ... H_n − 2 x_i^{(3)}
end do parallel
```

Figure 13: Parallel algorithm for finding eigenvectors

GEO-OK AM1 PRECISE
11-cis retinal crystal coords

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| C | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| C | 1.523 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| C | 1.498 | 1 | 112.1 | 1 | 0 | 1 | 2 | 1 | 0 |
| C | 1.532 | 1 | 111.9 | 1 | -59.1 | 1 | 3 | 2 | 1 |
| C | 1.521 | 1 | 113.3 | 1 | 39.8 | 1 | 4 | 3 | 2 |
| C | 1.333 | 1 | 122.9 | 1 | -11.5 | 1 | 5 | 4 | 3 |
| C | 1.486 | 1 | 114.0 | 1 | 180 | 1 | 6 | 5 | 4 |
| C | 1.339 | 1 | 126.2 | 1 | 41.4 | 1 | 7 | 6 | 5 |
| C | 1.461 | 1 | 126.4 | 1 | -179.6 | 1 | 8 | 7 | 6 |
| C | 1.347 | 1 | 117.8 | 1 | -174.0 | 1 | 9 | 8 | 7 |
| C | 1.454 | 1 | 125.3 | 1 | -175.5 | 1 | 10 | 9 | 8 |
| C | 1.339 | 1 | 128.1 | 1 | -179.3 | 1 | 11 | 10 | 9 |
| C | 1.472 | 1 | 129.9 | 1 | 2.1 | 1 | 12 | 11 | 10 |
| C | 1.358 | 1 | 121.3 | 1 | 38.7 | 1 | 13 | 12 | 11 |
| C | 1.467 | 1 | 122.9 | 1 | -179.8 | 1 | 14 | 13 | 12 |
| O | 1.213 | 1 | 121.4 | 1 | 174.5 | 1 | 15 | 14 | 13 |
| C | 1.500 | 1 | 125.8 | 1 | 3.9 | 1 | 5 | 6 | 7 |
| C | 1.563 | 1 | 106.9 | 1 | 42.8 | 1 | 1 | 6 | 7 |
| C | 1.528 | 1 | 109.7 | 1 | -74.3 | 1 | 1 | 6 | 7 |
| C | 1.511 | 1 | 117.9 | 1 | -7.2 | 1 | 9 | 8 | 7 |
| C | 1.526 | 1 | 124.7 | 1 | -3.1 | 1 | 13 | 14 | 15 |
| H | 1.000 | 1 | 110.0 | 1 | 55 | 1 | 4 | 3 | 2 |
| H | 1.000 | 1 | 110.0 | 1 | -55 | 1 | 4 | 3 | 2 |
| H | 1.000 | 1 | 110.0 | 1 | 55 | 1 | 3 | 2 | 1 |
| H | 1.000 | 1 | 110.0 | 1 | -55 | 1 | 3 | 2 | 1 |
| H | 1.000 | 1 | 110.0 | 1 | 55 | 1 | 2 | 3 | 4 |
| H | 1.000 | 1 | 110.0 | 1 | -55 | 1 | 2 | 3 | 4 |
| H | 1.000 | 1 | 120.0 | 1 | 0 | 1 | 7 | 8 | 9 |
| H | 1.000 | 1 | 120.0 | 1 | 0 | 1 | 8 | 9 | 10 |
| H | 1.000 | 1 | 120.0 | 1 | 0 | 1 | 10 | 9 | 8 |
| H | 1.000 | 1 | 120.0 | 1 | 0 | 1 | 11 | 10 | 9 |
| H | 1.000 | 1 | 120.0 | 1 | 180 | 1 | 12 | 11 | 10 |
| H | 1.000 | 1 | 120.0 | 1 | 0 | 1 | 14 | 13 | 12 |
| H | 1.000 | 1 | 120.0 | 1 | 0 | 1 | 15 | 14 | 13 |
| H | 1.000 | 1 | 110.0 | 1 | 60 | 1 | 18 | 1 | 2 |
| H | 1.000 | 1 | 110.0 | 1 | 120 | 1 | 18 | 1 | 35 |
| H | 1.000 | 1 | 110.0 | 1 | 240 | 1 | 18 | 1 | 35 |
| H | 1.000 | 1 | 110.0 | 1 | 60 | 1 | 19 | 1 | 2 |
| H | 1.000 | 1 | 110.0 | 1 | 120 | 1 | 19 | 1 | 38 |
| H | 1.000 | 1 | 110.0 | 1 | 240 | 1 | 19 | 1 | 38 |
| H | 1.000 | 1 | 110.0 | 1 | 60 | 1 | 20 | 9 | 8 |
| H | 1.000 | 1 | 110.0 | 1 | 120 | 1 | 20 | 9 | 41 |
| H | 1.000 | 1 | 110.0 | 1 | 240 | 1 | 20 | 9 | 41 |
| H | 1.000 | 1 | 110.0 | 1 | 60 | 1 | 21 | 13 | 12 |
| H | 1.000 | 1 | 110.0 | 1 | 120 | 1 | 21 | 13 | 44 |
| H | 1.000 | 1 | 110.0 | 1 | 240 | 1 | 21 | 13 | 44 |
| H | 1.000 | 1 | 110.0 | 1 | 60 | 1 | 17 | 5 | 4 |
| H | 1.000 | 1 | 110.0 | 1 | 120 | 1 | 17 | 5 | 47 |
| H | 1.000 | 1 | 110.0 | 1 | 240 | 1 | 17 | 5 | 47 |
| 0 | | | | | | | | | |

Figure 15: The input data for the Crystal to the MOPAC program

| Procedure | Description | Input | Output | Calling Procedure |
|---|---|---|---|---|
| DIAG | Rapid pseudo-diagonalization. Given a set of vectors which almost block-diagonalize a secular determinant, DIAG modifies the vectors so that the block-diagonalization is more exact. | Old vectors, Secular Determinant | New vectors | iter |
| DENSIT | Constructs the Coulson electron density matrix from the eigenvectors. | Eigenvectors, Number of singly and doubly occupied levels | density matrix | iter |
| ROTATE | All the two-electron repulsion integrals, the electron-nuclear attraction integrals, and the nuclear-nuclear repulsion term between two atoms are calculated. | atomic number of the first and second atom and their coordinates | two-electron repulsion integrals, electron-nuclear attraction integrals, nuclear-nuclear repulsion term | dhc, dhcore, hcore, solrot |
| FOCK2 | Adds on to Fock matrix the two-center two electron terms. | Fock matrix | (ITER) the entire Fock matrix is filled. (DERIV) only diatomic Fock matrices are constructed. | ITER and DERIV. |
| DIAT | Calculates overlap integrals between two atoms in general cartesian space. Principal quantum numbers up to 6, and angular quantum numbers up to 2 are allowed. | Atomic numbers and cartesian coordinates in Angstroms of the two atoms | Diatomic overlaps | H1ELEC |
| HQRII | Rapid diagonalization routine. | secular determinant | a set of eigenvectors and eigenvalues. The secular determinant is destroyed. | |
| REPP | Calculates the 22 two-electron reduced repulsion integrals, and the 8 electron-nuclear attraction integrals. These are in a local coordinate system. | atomic numbers of the two atoms, interatomic distance, and arrays to hold the calculated integrals. | two electron repulsion integrals, electron core attraction integrals | by ROTATE only |

Table 5: Some routines used in the calculations

| Procedure | Description | Input | Output | Calling Procedure |
|---|---|---|---|---|
| DCART | calculates the derivatives of the energy with respect to the cartesian coordinates. This is done by finite differences. | Coordinates | cartesian derivatives | deriv |
| DHC | Called by DCART and calculates the energy of a pair of atoms using the SCF density matrix. Used in the finite difference derivative calculation. It calculates only the energy contribution from those pairs of atoms that have been moved by deriv | no description | no description | dcart |
| JAB | Calculates the coulomb contribution to the Fock matrix in NDDO formalism | not described | not described | FOCK2 |
| COE | Within the general overlap routine COE calculates the angular coefficients for the s, p and d real atomic orbitals given the axis and returns the rotation matrix. | not described | not described | DENROT, DIAT |
| KAB | Calculates the exchange contribution to the Fock matrix in NDDO formalism. | not described | not described | FOCK2 |
| DIAT2 | Calculates reduced overlap integrals between atoms of principal quantum numbers 1, 2, and 3, for s and p orbitals. Faster than the SS in DIAT. This is a dedicated subroutine, and is unable to stand alone without considerable backup. | not discribed | not discribed | DIAT |
| GMETRY | Fills the cartesian coordinates array. Data are supplied from the array GEO, GEO can be (a) in internal coordinates, or (b) in cartesian coordinates. If STEP is non-zero, then the coordinates are modified in light of the other geometry and STEP. | internal coordinats | cartesian coordinates | HCORE, DERIV, READMO, WRITMO, MOLDAT, etc. |
| BINTGS | Calculates the B-functions in the Slater overlap. | not discribed | not discribed | |
| HELECT | Given the density matrix, and the one electron and Fock matrices, calculates the electronic energy. No data are changed by a call of HELECT. | density matrix, one and two electron matrix | electronic energy | ITER and DERIV |
| HCORE | generates the one-electron matrix and two electron integrals for a given molecule whose geometry is given in cartesian coordinates. | Coordinates of the molecule | one-electron matrix | two-electron integrals, nuclear energy |

Table 6: Some routines used in the calculations

iter generates a scf field and returns the energy the routine changes the following main data
*total density matrix, alpha density matrix,beta density matrix, eigenvectors, one-electron matrix*
*one-electron matrix, the fock matrix, the two-electron matrix*
- Generate a common block so that they can be used by the subrotines
- Make initalization dependent on keyword.
- Slightly perturb the density matrix in case the system is trapped in a s**2 = 0 state.
- Reset the density matrix if meci has formed an excited state. This prevents the scf getting trapped on an excited state, particularly if the pulay converger is used.
- do some other initializations

Start the scf loop here
- Make the alpha fock matrix
- Shift will apply to the virtual energy levels used in the pseudodiagonaliization.
- If the pseudodiagonalization approximation is invalid (the wavefunction is almost stable) set a switch
- If system goes unstable, limit shift but if system is stable do no limitations
- Do some update for the convergers
- Slightly perturb the fock matrix in case the system is trapped in a metastable excited electronic state O(n)
  - fock2(a,...), fock1 (a,...)
- make the beta fock matrix $O(no^2)$, fock2(b,...), fock1 (b,...)
- calculate the energy in kcal/mole O(1)
- make sure self-consistency test is not more stringent than the computer can handle
- self-consistency test for quick exit
  - invoke the camp-king converger
  - invoke pulay's converger
- diagonalize the alpha or rhf secular determinant
- where possible, use the pulay-stewart method, otherwise use beppu's dependent on the method some of the procedures mentioned bellow are used alternatively
  (hqrii(...) or diag (...))
- calculate the alpha or rhf density matrix
  densit(...)
  densit(...)
  does it converge?
- calculate the beta density matrix
  densit(...)
- calculate the total density matrix

end of the scf loop
- calculate the electronic energy
- normally the eigenvalues are incorrect because the pseudodiagonalization has been used. if this is the last scf, then do an exact diagonalization (hqrii)

Figure 14: outline of the procedure iter